Albertas Caplinskas
Johann Eder (Eds.)

LNCS 2151

Advances in Databases and Information Systems

5th East European Conference, ADBIS 2001
Vilnius, Lithuania, September 2001
Proceedings

Springer

Lecture Notes in Computer Science 2151
Edited by G. Goos, J. Hartmanis, and J. van Leeuwen

Springer
Berlin
Heidelberg
New York
Barcelona
Hong Kong
London
Milan
Paris
Tokyo

Albertas Caplinskas   Johann Eder (Eds.)

Advances in Databases and Information Systems

5th East European Conference, ADBIS 2001
Vilnius, Lithuania, September 25-28, 2001
Proceedings

Springer

Series Editors

Gerhard Goos, Karlsruhe University, Germany
Juris Hartmanis, Cornell University, NY, USA
Jan van Leeuwen, Utrecht University, The Netherlands

Volume Editors

Albertas Caplinskas
Institute of Mathematics and Informatics
Akademijos st. 4, 2600 Vilnius, Lithuania
E-mail: alcapl@ktl.mii.lt

Johann Eder
University of Klagenfurt, Department of Informatics Systems
Universitätsstr. 65, 9020 Klagenfurt, Austria
E-mail: eder@isys.uni-klu.ac.at

Cataloging-in-Publication Data applied for

Die Deutsche Bibliothek - CIP-Einheitsaufnahme

Advances in databases and information systems : 5th East European conference ; proceedings / ADBIS 2001, Vilnius, Lithuania, September 25 - 28, 2001. Albertas Caplinskas ; Johann Eder (ed.). - Berlin ; Heidelberg ; New York ; Barcelona ; Hong Kong ; London ; Milan ; Paris ; Tokyo : Springer, 2001
(Lecture notes in computer science ; Vol. 2151)
ISBN 3-540-42555-1

CR Subject Classification (1998): H.2, H.3, H.4, H.5, J.1

ISSN 0302-9743
ISBN 3-540-42555-1 Springer-Verlag Berlin Heidelberg New York

Typesetting: Camera-ready by author, data conversion by PTP-Berlin, Stefan Sossna
Printed on acid-free paper SPIN 10845509 06/3142 5 4 3 2 1 0

Preface

ADBIS 2001 continued the series of Eastern European Conferences on Advances in Databases and Information Systems. It took place in Vilnius, the capital of Lithuania.

The ADBIS series already has established some reputation as a scientific event of high quality serving as an internationally highly visible showcase for research achievements in the field of databases and information systems in the Eastern and Central European region as well as a forum for communication and exchange with and among researchers from this very area. The topics of the conference were related to the European Commission's initiative "eEurope – Information Society for All". They represent research areas that will greatly influence the functionality, usability, and acceptability of future information products and services. The conference sought to bring together all major players (experienced researchers, young researchers, representatives from industry), to discuss the state of the art and practical solutions, and to initiate promising future research. In order to ensure the active participation of representatives from industry the conference provided special industrial sessions.

The theme of ADBIS 2001 was set with invited talks given by renowned scientists covering important aspects of this wide field of research: Web access, processes and services in e-commerce, and content management.

The call for papers attracted 82 submissions from 30 countries. In a rigorous reviewing process the international program committee selected 25 papers for long presentation and inclusion in these proceedings. All these papers have been reviewed by at least three reviewers who evaluated their originality, significance, relevance, and presentation and found their quality suitable for international publication.

Topically, the accepted papers span a wide spectrum of the database and information systems field: from query optimization and transaction processing via design methods to application oriented topics like XML and data on the web.

Furthermore, the program featured two tutorials, presentations of short papers with challenging novel ideas at an early stage, experience reports, and industrial papers on the application of databases and information systems.

We would like to express our thanks and acknowledgement to all the people who contributed to ADBIS 2001:

- the authors, who submitted papers to the conference,
- the reviewers and the international program committee who made this conference possible by voluntarily giving of their time and expertise fto ensure the quality of the scientific program
- the sponsors of ADBIS 2001 who supported the organization of the conference and permitted the participation in particular of young scientists from Eastern countries
- all the good spirits and helpful hands of the local organization committee

- Mr. Günter Millahn (Brandenburg University of Technology at Cottbus) for maintaining the conference server
- the Springer-Verlag for publishing these proceedings and Mr. Alfred Hofmann for the effective support in producing these proceedings
- and last but not least we thank the steering committee and, in particular, its chairman, Leonid Kalinichenko, for their advice and guidance.

Finally, we hope that all who contributed to this event see this volume as a reward and as representation of scientific spirit and technical progress communicating exciting ideas and achievements.

June 2001 Albertas Caplinskas and Johann Eder

Conference Organization

The ADBIS 2001 Conference on Advances in Databases and Information Systems was organized by the Vilnius Gediminas Technical University, the Institute of Mathematics and Informatics, Vilnius, and the Lithuanian Computer Society in cooperation with the ACM Sigmod Moscow Chapter and the Lithuanian Law University.

General Chair

Edmundas Zavadskas (Vilnius Gediminas Technical University, Lithuania)

Program Committee Co-chairs

Albertas Caplinskas (Institute of Mathematics and Informatics, Lithuania)
Johann Eder (University of Klagenfurt, Austria)

Program Committee

Suad Alagic (Wichita State University, USA)
Leopoldo Bertosi (Pontificia Universidad Catoloca de Chile, Chile)
Juris Borzovs (Riga Information Technology Institute, Latvia)
Omran A. Bukhres (Purdue University, USA)
Wojciech Cellary (Poznan University of Economics, Poland)
Bohdan Czejdo (Loyola University, USA)
Hans-Dieter Ehrich (Braunschweig Technical University, Germany)
Heinz Frank (University of Klagenfurt, Austria)
Remigijus Gustas (University of Karlstad, Sweden)
Tomas Hruska (Brno Technical University, Slovakia)
Yoshiharu Ishikawa (University of Tsukuba, Japan)
Leonid Kalinichenko (Institute for Problems of Informatics, Russian Academy of Sciences, Russia)
Wolfgang Klas (Ulm University, Germany)
Matthias Klusch (German Research Center for Artificial Intelligence GmbH, Germany)
Mikhail R. Kogalovsky (Market Economy Institute, Russian Academy of Sciences, Russia)
Kalle Lyytinen (University of Jyvaskyla, Finland)
Yanis Manolopoulos (Aristotle University, Greece)
Michail Matskin (Norwegian University of Science and Technology, Norway)
Tomaz Mohoric (Lublijana University, Slovenia)
Tadeusz Morzy (Poznan University of Technology, Poland)
Pavol Navrat (Slovak University of Technology, Slovakia)

Additional Referees

Organizing Committee

Chair

Olegas Vasilecas (Vilnius Gediminas Technical University, Lithuania)

Vice-Chairs

Algimantas Ciucelis (Vilnius Gediminas Technical University, Lithuania)
Alfredas Otas (Lithuanian Computer Society, Lithuania)
Rimantas Petrauskas (Lithuanian Law University, Lithuania)

Members

Petras Adomenas (Vilnius Gediminas Technical University, Lithuania)
Danute Burokiene (Institute of Mathematics and Informatics, Lithuania)
Dale Dzemydiene (Institute of Mathematics and Informatics, Lithuania)
Milda Garmute (Vilnius Gediminas Technical University, Lithuania)
Audrius Klevas (Vilnius Gediminas Technical University, Lithuania)
Kristina Lapin (Vilnius University, Lithuania)
Audrone Lupeikiene (Institute of Mathematics and Informatics, Lithuania)
Saulius Maskeliunas (Institute of Mathematics and Informatics, Lithuania)
Guenter Millahn (Brandenburg University of Technology at Cottbus, Germany)
Arunas Ribikauskas (Vilnius Gediminas Technical University, Lithuania)
Danute Vanseviciene (Institute of Mathematics and Informatics, Lithuania)
Aldona Zaldokiene (Institute of Mathematics and Informatics, Lithuania)

ADBIS Steering Committee

Chair

Leonid Kalinichenko (Russia)

Members

Andras Benczur (Hungary)
Radu Bercaru (Romania)
Albertas Caplinskas (Lithuania)
Johann Eder (Austria)
Janis Eiduks (Latvia)
Hele-Mai Haav (Estonia)
Mirjana Ivanovic (Yugoslavia)
Mikhail Kogalovsky (Russia)
Yannis Manopoulos (Greece)
Rainer Manthey (Germany)
Tadeusz Morzy (Poland)
Pavol Navrat (Slovakia)
Boris Novikov (Russia)
Jaroslav Pokorny (Czech Republic)
Boris Rachev (Bulgaria)
Anatoly Stogny (Ukraine)
Tatjana Welzer (Slovenia)
Viacheslav Wolfengagen (Russia)

Sponsoring Institutions

European Commission, Research DG, Human Potential Programme,
High-level Scientific Conferences (subject to contract)
Lithuanian Science and Studies State Foundation
Microsoft Research Ltd.

Table of Contents

Ubiquitous Web Applications

Franca Garzotto

HOC- Hypermedia Open Center
Department of Electronics and Information
Politecnico di Milano, Italy
garzotto@elet.polimi.it

Abstract. Web sites are progressively evolving from browsable, read-only information repositories that exploit the web to interact with their users, to web-based applications, combining navigation and search capabilities with operations and transactions typical of information systems. In parallel, the possibility of accessing web-based contents and services through a number of different devices, ranging from full-fledged desktop computers, to Personal Digital Assistants (PDA's), to mobile phones, to set-top boxes connected to TV's, makes web applications ubiquitous, i.e., accessible anywhere at any time.
Ubiquitous Web Applications (UWA's for short) reveal a number of aspects which make them different with respect to a conventional data-intensive applications, and must be taken into account throughout the whole application lifecycle, from requirements to implementation. UWA's are executed in a Web-based environment, where the paradigm for presenting and accessing information is hypermedia-like. Thus UWA's have a mixed nature - hypermedia and transactional, where hypertext structures and operation capabilities are strongly intertwined. In addition, the ubiquitous nature of a UWA implies that the application has to take into account the different constraints of different devices, comprising display size, local storage size, method of input and computing speed as well as network capacity. At the same time, ubiquity introduces new requirements on how the application tunes itself to the end user: each user may wish to get information, navigation patterns, lay-out, and services, that are tailored not only to his/her specific profile but also to the current situation of use, in its temporal and environmental aspects. Thus Ubiquitous Web Applications must be at the same time device-aware, user-aware, and context-of-use-aware, and require sophisticate forms of customization.
After an analysis of the novel requirements of UWA's, this talk will focus on their impact on the design process, and will discuss problems and challenges related to modeling information and navigation structures, operations and transactions, and customization mechanisms for this class of applications.

From Workflows to Service Composition in Virtual Enterprises

Marek Rusinkiewicz

Information and Computer Science Research Laboratory
Telcordia Technologies
Austin, TX, USA
marek@research.telcordia.com

Abstract. Workflow technologies have been used extensively to support process-based integration of activities within enterprises and are at the core of emerging Enterprise Integration Platform (EIP) technologies. Recently, a new abstraction of electronic services has begun to receive a lot of attention among researchers and in the vendor community. Electronic services provide the basis for creation of virtual enterprises (VE), which combine services from multiple providers. In this talk, we will discuss the advances that are needed to provide support for VEs. These include the ability to advertise, broker, synchronize and optimize the services. We will discuss the infrastructure needed to support VE application, show a prototype demo, and list important research problems that need to be solved.

Subject-Oriented Work: Lessons Learned from an Interdisciplinary Content Management Project

Joachim W. Schmidt, Hans-Werner Sehring, Michael Skusa, and Axel Wienberg

Technical University TUHH Hamburg, Software Systems Institute,
Harburger Schloßstraße 20, D-21073 Hamburg, Germany
{j.w.schmidt,hw.sehring,skusa,ax.wienberg}@tuhh.de

Abstract. The two broad cases, data- and content-based applications, differ substantially in the fact that data case applications are abstracted first before they cross any system boundary while for content cases it is the system itself which has to map application content into some data-based technology. Through application analysis and software design we are aware of the difficulties of such mappings. In an interdisciplinary project with our Art History colleagues who are working in the subject area of “Political Iconography” we are gaining substantial insight into their *Subject-Oriented Working* (SOWing) needs and into initial requirements for a SOWing environment. In this paper we outline the project, its basic models, their generalization as well as our initial experiences with prototypical SOWing implementations. We emphasizes the conceptual and terminological aspects of our approach, sketch some of the technical requirements of a generic SOWing software platform and relate our work to various XML-based activities.

1 Introduction

As a result of advanced and extensible database technology now being available as off-the-shelf products, a substantial part of database research and development work has generalized into work on models and systems for multimedia content management. R&D in content management includes a range of models and systems concentrating on services for the following three lines of work:

- content production and publication work using multimedia documents;
- classification and retrieval work based on document content;
- management and control of such work for communities of users differentiated by their roles and rights, interests and profiles, etc.

The work reported in this paper is based on an interdisciplinary project with a partner from the humanities with strong semantic and weak formal commitment. Our project partner specializes on work in icon- and text-based content from the subject area of “Political Iconography”. This content is organized as a paper- and drawer-based subject index (PI-“Bildindex”, BPI, see Fig. 1) and is used for Art History research and education [33].

Iconographic work has a long tradition in Art History, dating back to 19th century “Christian Iconography”, and is based on integrated experience from three sources:

- Art: multimedia content;
- Art History: process knowledge;
- Library Sciences: subject-oriented content classification and retrieval.

In our context we use the notion of *subject-orientation* very much in the sense of library science, as, for example, stated by Elaine Svenonius: "In a subject language the extension of a term is the class of all documents about what the term denotes, such as all documents about butterflies." This understanding differs substantially from natural language where "the extension, or extensional meaning, of a word is the class of entities denoted by that word, such as the class consisting of all butterflies" [30]. And both understandings are in clear contrast to the semantics of terms in programming languages and database models.

The interdisciplinary project "Warburg Electronic Library (WEL)" models and computerizes BPI content and services [3], [19], and the WEL prototype allows interdisciplinary experiments and insights into multimedia content management and applications.

The overall goal of the WEL project is
- the generalization of our subject-oriented working experience,
- a work plan for R&D in subject-oriented content management, and
- a generic Subject-Oriented Working environment (SOWing environment).

Currently, many contributions to such R&D are based on XML as a syntactic framework which provides a structural basis as well as some form of implementation platform. The main reasons for XML's powerful position are its strong structural commitment and its semantic neutrality.

Successful content management requires that the three lines of work
- content production and publication work by multimedia documents;
- classification and retrieval work based on document content;
- management and control of such work for communities of users differentiated by their roles an rights, profiles and interests, etc. [21]

are not supported in isolation but in a coherent and cooperative working environment covering the entire space spanned by all three dimensions.

The main reason why XML-based work on content management often falls short can be stated as a corollary of XML's strength (see above): its weak semantic and exclusively structural commitment. Much of the XML-based R&D contributes to the above three lines of work only individually. Examples include [7], [31].

The paper is structured as follows. Section 2 introduces the two projects involved, the "Bildindex für Politische Ikonographie (BPI)" and the "Warburg Electronic Library (WEL)". In Sect. 3 the WEL model is generalized towards a generic "Work Explication Language". A system's view of the WEL prototype is described in Sect. 4 and the first contours of a generic Subject-Oriented Working environment (SOWing platform) are outlined. Related work, in particular work in the XML context, is discussed in Sect. 5. The paper concludes with a short summary and a reference to future work in our long-term SOWing project.

2 Warburg Electronic Library: An Interdisciplinary Content Management Project

The development of the currently predominant data management models was heavily influenced by application requirements from the business and banking world and their bookkeeping experience: the concepts of record, tabular view, transaction etc. are obvious examples. Data model development had to go through several generations – record-based file management, hierarchical databases and network models – until the relational data model reached a widely accepted level of abstraction for database structuring and content-based data operation.

For traditional relational data management we basically assume that content is "values of quantified variables" from business domains operated by transactions and laid out as tables with rows and columns. The questions arise:

- How can we generalize from data management to the area of content management where content domains are not "just dates and dollars", content operation goes beyond "debit-credit transactions" and content layout means multimedia documents?
- What are key application areas beyond bookkeeping which help us understand, conceptualize and finally implement the core set of requirements for multimedia content management in terms of domain modelling, content-oriented work support as well as content (re-) presentation?

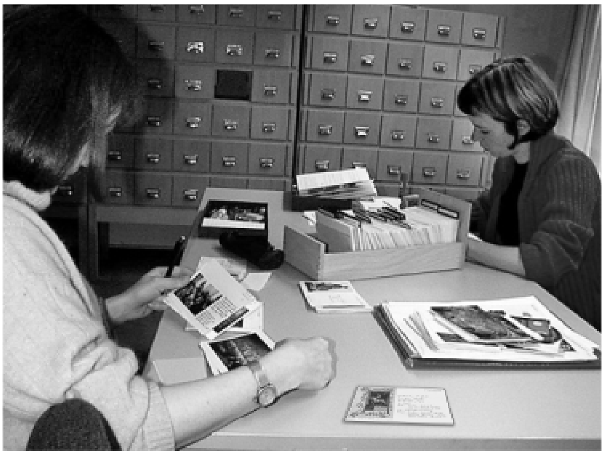
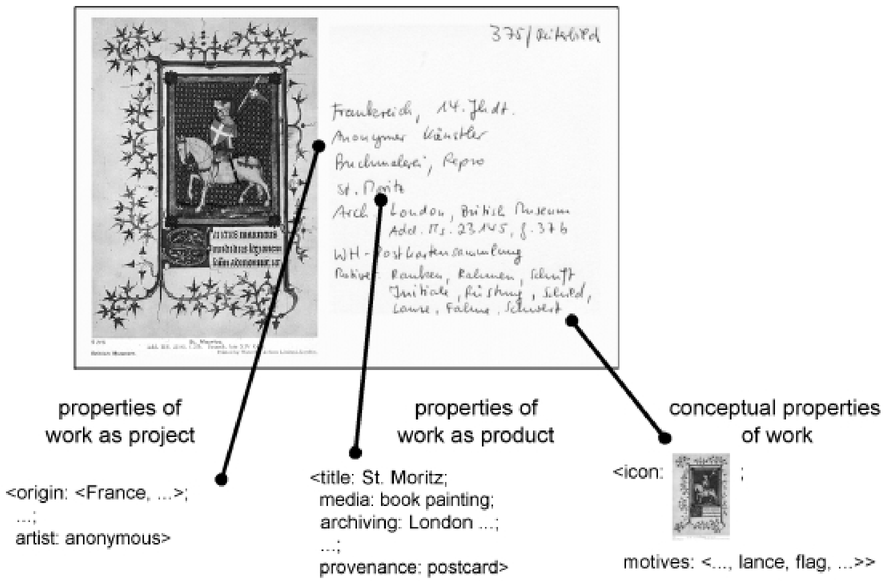


Fig. 1. Working with the Index for Political Iconography in the Warburg-Haus, Hamburg

The work presented here is based on an interdisciplinary R&D-project between Computer Science and Art History, the "Warburg Electronic Library" project. The application area was chosen because of Art History's long-term working experience with content of various media. The project itself is founded on extensive material and user experience from the area of "Political Iconography".

2.1 Subject-Oriented Work in Political Iconography

Political iconography basically intends to capture the semantics of key concepts of the political realm under the assumption that political goals, roles, values, means etc. requires mass communication which is implemented by the iconographic use of images. Our partner project in Art History, the "Bildindex zur Politischen Iconographie (BPI)", was initiated in 1982 by the Art Historian Martin Warnke [33] and consists of roughly 1,500 named political concepts (*subject terms*, "Schlagworte") and more than 300,000 records on iconographic works relevant to the BPI. In 1990 Warnke's work was awarded the Leibniz-Preis, one of the most prestigious research grants in Germany.



properties of work as project
<origin: <France, ...>; ...; artist: anonymous>
properties of work as product
<title: St. Moritz; media: book painting; archiving: London ...; ...; provenance: postcard>
conceptual properties of work
<icon: ;
motives: <..., lance, flag, ...>>

Fig. 2. BPI "Bildkarte" St. Moritz (image card) describing art work by attribute aggregation

Starting with this experience, BPI work essentially relies on an Art Historian's knowledge of (documents refering to) political acts in which images play an active role. Art Historians interpret "acts" as encompassing aspects of

- "projects" (who initiated and contributed to an act? the when and where of an act? etc.);
- "products" (what piece of art did the project produce? on what medium? place of current residence etc.); and finally, the
- "concepts" behind the act (what political goals, roles, institutions etc. are addressed? what iconographic means are used by the artist? etc.).

On this knowledge level, BPI work identifies political concepts and names them individually by *subject terms* – e.g., by "ruler", "prince", "pope", "equestrian statue".

Subject term semantics is methodologically captured and systematically represented in the BPI by the following steps:

1. designing a conceptual, prototypical and representative (mostly mental) model for each subject term, e.g., a prototypical equestrian statute; [29]
2. giving value to the relevant variables or facets of such prototypes by reference to the Art Historian's knowledge of "good cases", i.e., political acts with an iconographic dimension. Each such variable or facet is represented by a BPI entry ("Bildkarte", "Textkarte", "Videokarte" – "media card" etc.) which holds a description of a "good case" for that facet, see for example, St. Moritz, see Fig. 2.
3. collecting all BPI entries on the same prototype into a single extent ("Bildkartenstapel", …, "stack of media cards", see Fig. 3) thus defining the semantics of a subject term. Additional fine structure may be imposed on subject term extents (order, "neighborhood", named subextents, general association/navigation etc.);
4. maintaining a ("completion") process aiming at a "best possible" definition of the subject area at hand by
    - "representative" subject terms covering the subject area at hand;
    - "qualifying" prototypes for each subject term;
    - "complete" sets of facets for prototype description;
    - "good" cases for facet substantiation.

This makes it quite clear that the BPI is by no means just an index for accessing an image repository. The BPI uses images only in their rather specific role as icons and for the specific purpose of contributing to the description of cases and thus to the semantics of subject terms [32]. In this sense, images represent the iconographic vocabulary of BPI documents just as keywords contribute to the linguistic vocabulary of text documents.

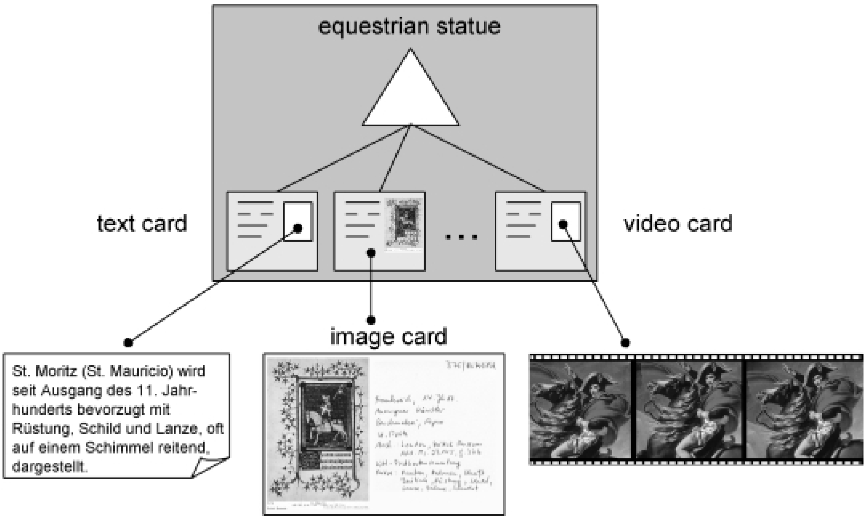

equestrian statue
text card
video card
image card
St. Moritz (St. Mauricio) wird seit Ausgang des 11. Jahrhunderts bevorzugt mit Rüstung, Schild und Lanze, oft auf einem Schimmel reitend, dargestellt.

Fig. 3. BPI subject term semantics (e.g. equestrian statue) by media card classification (image, text, video cards etc.)

Art Historians with their long tradition of working with content represented by multiple media are far from restricting themselves to a mainly technical view on multimedia as most of the currently booming projects in Computer Science seem to do. Our Art History colleagues are much closer to the message of people such as Marshall McLuhan who understand media as “extensions of men [18]”.

The BPI has essentially two groups of users:
- a few highly experienced BPI editors for content maintenance and
- various broader user communities which access BPI content for research and education purposes.

Being implemented on paper technology, the traditional BPI shows severe conceptual and technical shortcomings:
- conceptually: the above attributes “representative”, “good”, “complete”, etc. are highly subjective and, therefore, “completion semantics” is hard to meet even within a “single-person-owned” subject index;
- technically: severe representational limitations are obvious and range from single subsumption of BPI entries to a lack of online and networked BPI access.

In the subsequent section we outline two contributions of the “Warburg Electronic Library” project which approaches the above conceptual and technical shortcomings through an advanced Digital Library project which, as a prime application, is now hosting the “Index for Political Iconography”.

2.2 A Subject-Oriented Working Environment: Warburg Electronic Library

Viewed from our Computer Science perspective which shifted in recent years from basic research in "persistent database programming" towards R&D in "software systems for content management (online, multimedia, …)", the WEL project addresses a range of highly relevant and interrelated content application issues:

- content representation by multiple media: images, texts, data, …;
- content structuring, navigation and querying; content presentation;
- content work exploiting subjects and ontologies: classification, indexing, …;
- utilization of different referencing mechanisms: icon, index, symbol;
- cooperative projects on multimedia content in research and education.

The WEL is an interdisciplinary project between the Art History department of Hamburg University (Research Group on Political Iconography, Warburg-Haus, Hamburg) and the Software Systems Institute of the Technical University, TUHH, Hamburg. It began in 1996 as a 5-year project and will be extended into an interdisciplinary R&D-framework involving several Hamburg-based institutions.

For a short WEL overview we will concentrate on two project contributions:

- semantic modelling principles for WEL-design;
- personalized digital WEL libraries based on project-specific prototypes and their use in Art History education.

WEL Semantic Modelling Principles. The WEL design is based – as is already the BPI design – on the classical semantic data modelling principles [28], [6]: aggregation, classification, generalization / specialization and association / navigation (see figs. 2 and 3).

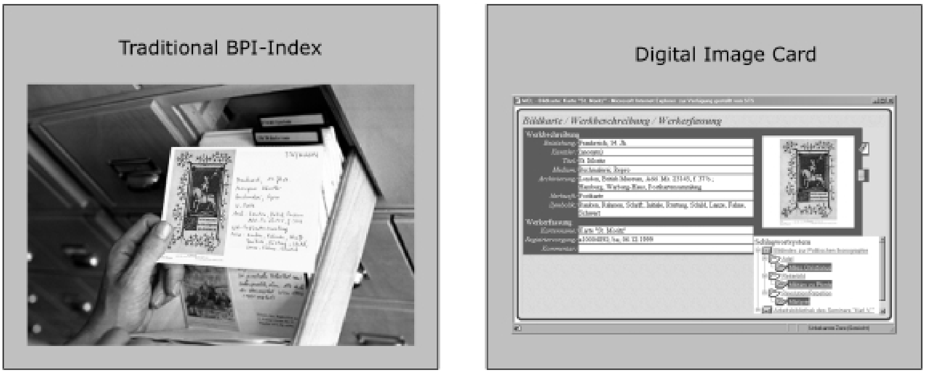


Fig. 4. Media card associated with (multiple) subject terms and with information on classification work

However, it is important to note that the semantics of subject classes and their entries originate from different semantic sources and, therefore, go beyond classical data modelling (see also Sect. 4.2):

- *object semantics*: seen from a data modelling point of view, subject class entries are also entities of some object classes in the sense of object-oriented modelling. However, a subject class extent may be heterogeneous because its entries may describe documents of different media – texts, images, videos etc. Therefore, subject class entries viewed as objects may belong to different object classes – text, image, video classes etc.
- *content semantics*: furthermore, all content described by the entries of the same subject class shares some semantic key elements. All BPI documents referring, for example, to the subject class "ruler" make use of graphical textual key icons such as swords, crowns, scepters, horses etc.; similarly, the text documents associated with a certain subject class contain overlapping sets of subject-related keywords. Such sets of key icons capture essential parts of content and, thus, of subject term semantics. Note that specialization of subject classes goes along with extension and union of subject-related key icon sets while generalization relies on reduction and intersection.
- *completion semantics*: in Sect. 2.1 we referred to the soft semantic constraint of achieving best possible subject definition as "completion semantics". Although this may be considered more as an issue of class *pragmatics* than class semantics it implies formal constraints on subject class extents. Since users of subject definitions act under the assumption that the subject owner established a subject extent which represents *all* relevant aspects of the owners subject prototype, any change of that extent is primarily monotonic, i.e., extents of subject terms are only changed by adding or replacing its entries. Therefore, references to subject class entries should not become invalid.

Figure 4 shows a media card for St. Moritz together with the (multiple) subject terms to which St. Moritz contributes. The example also links St. Moritz to details of the classification process by which this card entered the WEL. This information is essential for the realization of project-oriented views on subject terms, or reference libraries:

- thematic views (*customization*): projects usually concentrate on sub-areas of the all encompassing "Index for the Political Iconography";
- personalized views (*personalization*): they cope with the conceptual problems with "completion semantics" mentioned above.

Initial experiences with both of these viewing mechanisms are outlined in the subsequent section.

Subject-Oriented Work in Art History Education. A key experience of the WEL project relates to the two dimensions of subject-oriented work: subject-orientation as a thematic view (*customization*) and as an individualized view (*personalization*). Speaking in terms of Digital Libraries both dimensions are approached by the WEL concept of “reference libraries” (“Handbibliothek”), which are essentially SOWing environments customized and personalized according to the requirements of individual projects or persons [17].

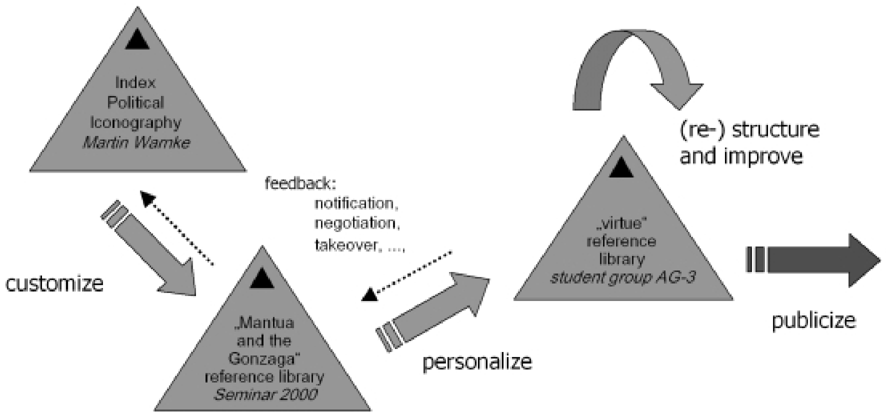


Fig. 5. Thematic and personalized subject views for the “Mantua” seminar

Figure 5 outlines the use of customized and personalized SOWing environments in an Art History seminar on “Mantua and the Gonzaga” [26]. The general BPI (“Warnke-owned”) is first customized into a subject index for the “Mantua and the Gonzaga” seminar project. The main objective of the individual student projects in that seminar is to further personalize the seminar index, structurally and content-wise, and to produce, for example, a project-specific subject index for topics such as “Studiolo” or “Camera picta” [20]. Publicizing the final subject content in some form of media document (see Fig. 6) - traditional print report or interactive website - constitutes another seminar objective [34].

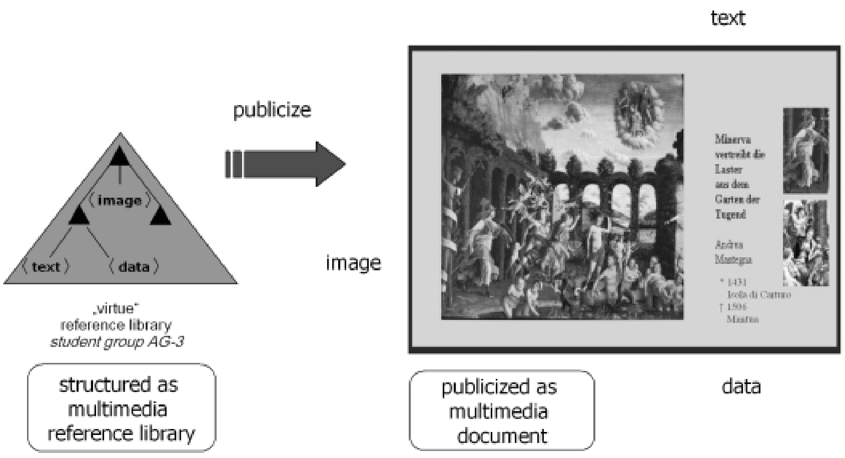

text
publicize
image
image
text
data
„virtue"
reference library
student group AG-3
structured as multimedia reference library
Minerva vertreibt die Laster aus dem Garten der Tugend
Andrea Mantegna
* 1431 Isola di Cartaro
† 1506 Mantua
publicized as multimedia document
data

Fig. 6. The "Studiolo" subject index publicized as a multimedia document

3 Towards a Generalized WEL-Model for Subject-Oriented Work

The prime experience gained from our interdisciplinary WEL project is a deeper insight into the intertwining of SOWing entities and their working relationships. Conceptually, the services of our SOWing platform are primarily based on four kinds of entities which are straight-forward generalizations of the corresponding WEL entities:

- "work cases" as the basic abstraction of the acts and entities of interest in a domain;
- "case documents" which are abstract or physical entities reporting on such work;
- "case entries" which record the essence of work case documents;
- "subject terms" which, based on such case entries, structure the domain, define its semantics, and give access to its documents and works.

Since all four notions are quite generic, there is ample space for generalizing in our SOWing project the models and systems for subject-oriented work far beyond our initial WEL approach.

In the subsequent sections we will outline an extended SOWing model and platform and re-interpret the acronym WEL from "Warburg Electronic Library" to "Work Explication Language", very much in the sense of the definition of ontology as "a theory regarding entities, especially abstract entities to be admitted into a language of description" [35].

Subsequently, we discuss the four kinds of SOWing entities in more detail.

3.1 On a Generic Notion of "Work"

Central to the content management support provided by our SOWing platform is a generic notion of "work". Our work concept is based on the WEL experience and is characterized and modeled by three groups of properties (see also Fig. 2):

- work as a "project", i.e., the circumstances under which work is performed;
- work as a "product", i.e., a work's result; and
- work as a "concept", i.e., the conceptual idea behind a work.

Figure 7 relates these three work characteristics using a "work triangle" diagram. The generalization of WEL work examples such as the one given by Fig. 2 is obvious.

The upper part of Fig. 8 depicts a second work case, similarly structured but rather different in nature. Fig. 8.2 depicts a WEL work case done by a Mr. B. when producing a work document description of the Gonzaga-Mantegna-Minerva work case and entering it into the WEL. While there may be only partial knowledge on the renaissance work case – essentially only Mantegna's picture survived – the WEL work case being supported and observed by the SOWing platform can receive an arbitrarily extensive SOWing coverage.

This reflective capability is probably the most powerful and unique aspect of our SOWing approach. Reflection provides the basis for a wide range of services for customization and personalization, self-description and profiling and for all kinds of guiding and tracing support [27].

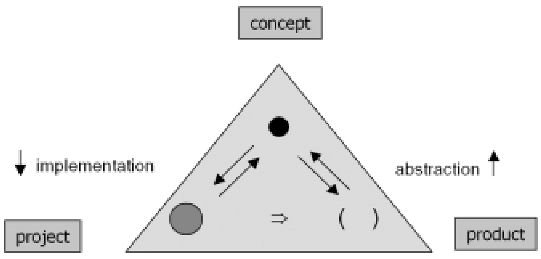


Fig. 7. WEL work structure (work triangle)

Examples of WEL work are presented in Fig. 8. The lower part (Fig. 8.1) models a specific work case by which a member of the Gonzaga family residing in Mantua during the 15th century asked the artist Mantegna for a painting addressing the issue of virtues and sins. Mantegna chose the goddess Minerva as the central motive and presented her expelling the sin out of the garden of virtue. This 15th century work case may be reported by some publicized work document, most probably, however, the case is just part of an Art Historian's body of knowledge about the Italian renaissance.

3.2 Work Case Documents

Work such as the Gonzaga-Mantegna-Minerva case of Fig. 8 is documented typically in narrative form and presented by some multi-media documents – texts, images, speech etc. and combinations thereof. In our SOWing approach such documents are assumed to represent content in terms of the above three dimensions: work project, work product and the concepts behind both (see also Fig. 7). This view on documents is quite general and allows interpretation ranging from the rather informal but very expressive documents of "Political Iconography" to partially formalized diagrams such as flow charts and UML entities and to computer programs and operator instructions with a fully formalized semantics.

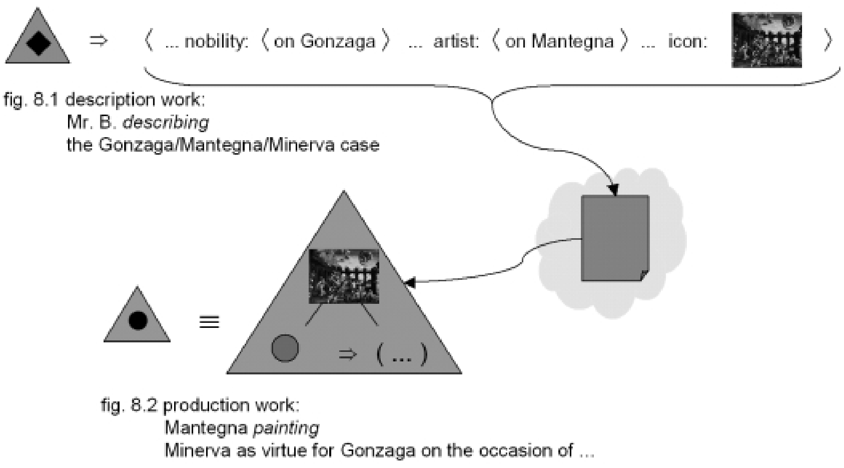

⇒ 〈 ... nobility: 〈 on Gonzaga 〉 ... artist: 〈 on Mantegna 〉 ... icon: 〉
fig. 8.1 description work:
Mr. B. *describing*
the Gonzaga/Mantegna/Minerva case
≡
⇒ (...)
fig. 8.2 production work:
Mantegna *painting*
Minerva as virtue for Gonzaga on the occasion of ...

Fig. 8. WEL work graph: production work (Fig. 8.1) and description work (Fig. 8.2)

Work documents may also vary in terms of their completeness in the sense that the work they report may be known only partially, for example, by its product. Many examples can be found in Art History where most of the project knowledge is usually lost and only the image survives (the opposite case also exists). Nevertheless, we agree with our colleagues from Art History that products should never be considered in isolation but always be recorded in the context of the project (persons, time, place, tools, etc.) for which they were created. For documents produced within a computerized environment this has partially become standard although there is no overall concept of what to do with this information.

Note the dialectic character of our SOWing position in this point: on the one hand side, work cases are assumed to be reported by documents, on the other hand all documents – at least the ones produced by the SOWing environment and its tools – are considered as work cases and, yet again, reported by work case documents and entries.

3.3 Work Case Entries

In the SOWing approach work case documents are described and recorded by work case entries which establish the relationship between subject terms and their semantics on one side and the work cases and their documentation on the other. Such case entries generalize the media cards of the WEL system.

Conceptually there is a third relationship involved which associates a work case entry with a class of icons considered “representative” for the kind of work which is described by the case. We use the term “icon” here in the sense of iconic signifier (as opposed to indexical or symbolic signifiers, see, for example, [9]). For image documents icons specialize to iconographic signifiers, for text documents to keywords (“Stichworte”) etc.

A case entry on an image and text document with content on the subject term “ruler”, for example, will for its image part be described by characteristic icons from a subject-specific icon set {crown, sword, scepter, ...}, for its textual part by a corresponding set of keywords.

Seen from an *object* point of view, case entries are instances of media-specific object classes (image, text, video classes etc.) while from a *content* perspective they draw constraints from (hierarchies of) icon classes. Finally, from a *subject*-oriented position, case entries become members of the extent of some subject classes (our old WEL stack of media cards) thus contributing to the definition of their semantics.

In Sect. 4 we will relate and discuss these three perspectives in terms of class diagrams (Fig. 10).

3.4 Subject Definition Work

Subject term semantics is essentially defined extensionally by document descriptions. Intentionally their semantics is captured in part by icon classes shared by such extents. Both extensional and intensional semantics are related by the fact that each entry into the subject term’s extent shows a characteristic profile over the icon class related to its subject term. A specific image document contributing to the semantics of “ruler” will not display the entire icon class for rulers, i.e., the set {crown, sword, scepter, ...} but a characteristic subset of it, probably a crown and a sword in a prominent position within the image.

Most of the production work on which the BPI is based typically took place outside a computerized environment – usually the referenced iconographic work dates back several centuries. Production work may, however, also mean the production of documents *about* the original iconographic work and such document work may well contribute to or profit from a SOWing environment.

Description support being definitely a matter of the SOWing platform may, for example, provide reference to other subject terms covered by the index [22]. For the Gonzaga-Mantegna-Minerva case it may be quite enlightening to capture the reason *why* Gonzaga ordered that picture by referring to some subject term “virtue” or its generalization “political objectives” to which Machiavelli’s work “Il Principe”, a successful handbook for renaissance rulers, contributed.

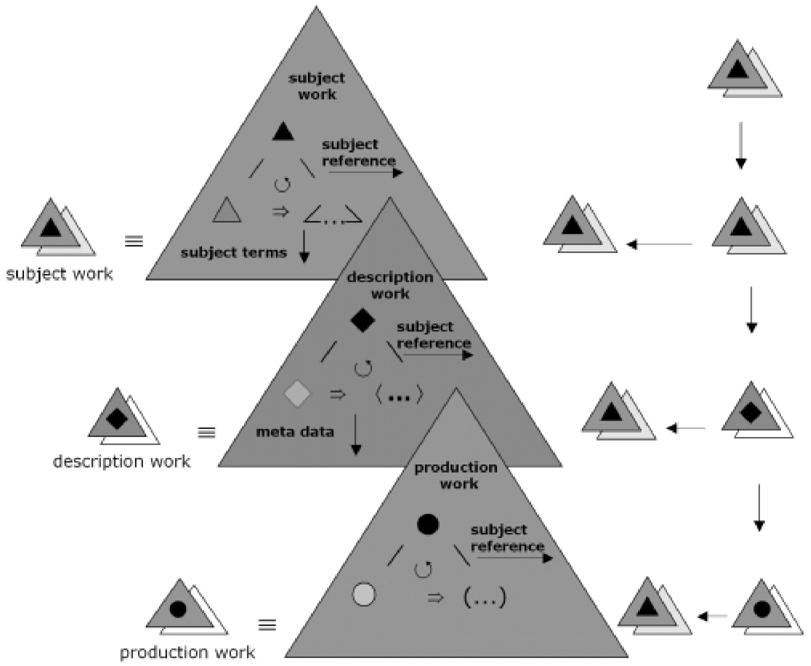


Fig. 9. An overview of Subject-Oriented Work

Figure 9 gives an overview of the subject-oriented work and its support through a SOWing platform. It relates production and description work to subject work.

As mentioned above, a major group of SOWing services is based on the fact that "work is a first class citizen" in the SOWing world. This implies that work while being supported by the system is automatically identified, described and associated with work-related subject terms (project time and participants, product media and archives etc.). Evaluating such terms provides the basis for substantially improved work support, e.g., work session management, work distribution, protection, personalization etc.

4 The SOWing System

Our field studies provided us with a good basis for user requirements analysis of our SOWing approach and our extensive prototyping experience allowed us us a deeper insight into the architectural and functional alternatives of SOWing system design and implementation.

4.1 WEL Prototype Experience

The WEL prototypes developed so far are the major source of experience on which the SOWing approach is based. The initial version was based on the Tycoon-2 persistent programming environment, an object-oriented, higher-order programming language with orthogonal persistence [16], [23]. Using a Tycoon-based acquisition and cataloging tool the Computer Scientists, Art Historians and many students from both departments digitized many thousands of images and transformed file cards from their physical representation into a digital one. This work is still in progress. The descriptive data in the card catalogue were entered and revised by Art Historians via a web-based editor.

With the evolution of Java from a small object-oriented programming language for embedded devices to a mainstream programming language for networked applications the WEL system was moved to Java-related technology.

The current WEL version is based on an early version of a commercial Content Management System (CMS) [5] which is entirely Java-based and resides on top of a relational database management system.

For our current prototype we were particularly interested in understanding to what extent features of commercial CMS technology meet the requirements of our SOWing platform. It turned out that although the CMS provided several abstractions useful for our SOWing data model important relationships between and inside the subject classes could not be treated as first class objects. Furthermore, the modeling facilities of commercial CMSs are not yet rich enough to meet the needs of our generic SOWing approach. Commercial CMS technology concentrates on specific application domains with editorial processes for more or less isolated units of work (e.g., news articles) with little association to other entities. SOWing requires, however, in addition to storing and retrieving the document itself an extended functionality for the embedding of documents into the SOWing context with all its relationships.

Commercial CMS and DBMS technology supports only the specific set of navigation methods predominant in their main application domain, and our prototype ran into performance problems as soon as users left those default navigation paths. As a consequence, we met the domain specific access requirements by introducing an additional application layer on top of the CMS.

Several graphical web-based user interfaces as well as GUI editors were developed to enable various classes of users to browse through the subject index, create personal indices and collect references to documents accessible via the global index.

In parallel to the Art History project the generic WEL system was adopted to maintain an index created for the management of concepts relevant in advertising. This was and still is carried out in cooperation with a commercial agency from the advertising industry.

4.2 Subject Classes, Object Classes, and Icon Classes

In Sect. 3 we introduced the four kinds of SOWing entities: work cases, case documents, case entries, and subject terms. While the former two (Sects. 3.1 and 3.2) are highly relevant for the conceptual foundation of our SOWing approach, the latter two (Sects. 3.3 and 3.4) are also important for system implementation.

Case entries are related to subject terms by classification relationships. Subject terms are structured in a hierarchy – the subject index – made up by subject term generalization and specialization.

As described in Sect. 2.1 the relationship between case entries and subject terms has a double meaning:

- one the one hand, documents are classified by binding their case entries to subject terms;
- on the other hand, each document contributes to the subject term's definition.

The set of entries chosen to define a subject is supposed to be minimal; only entries which introduce a new and relevant facet of the prototypical extent are added. In this way subject indices reflect the domain knowledge from the perspective of the owner of the subject index.

Subject indices are not only used to capture "primary knowledge" of domains but also "secondary knowledge" as, for example,

- on the organizational and history of a community, including knowledge of users, rights granted to them, etc. [8]
- on the layout and handling of documents, i.e., properties not directly related to their content, e.g., (kind of) origin, document types, quality, etc.

This leads to different types of subject indices some of which are used by the SOWing system itself, e.g., user classification to handle project-specific access rights.

Since entries can contribute to more than one subject term definition they may belong to more than one subject extent maybe in different indices of the same or different types. The semantics of multiple subsumption varies in each of the cases.

Several contributions define the semantics of the instantiation relationship:

- object semantics: technically, descriptions have a type (in the sense of a data type);
- content semantics: in addition to the attributes with object semantics, the use of icons for content description determines a semantic type; the more special a subject becomes the more icons the entries in its extent are expected to have;
- completion semantics: the extent of a subject term is supposed to fully describe its semantics (at the current time, for the user who created it).

It is interesting to see how in our design of SOWing entities traditional elements of object-oriented classes coexist with novel aspects of subject-oriented modelling. This coexistence of object- and subject-oriented semantic elements is illustrated by the diagram in Fig. 10 which is based on an extended UML notation.

The upper right third of the diagram shows a traditional class hierarchy of an object-oriented model. On top is the class "Object" representing the root of the class hierarchy. From this a class "Case Entry" is derived which has attributes for the features which all kinds of case entries share. Subclasses of "Case Entry" are introduced for each media type. These classes might introduce further features, e.g., painter for images, author for texts. Instances ("entry" in the diagram) of such classes are constructed in the usual object-oriented manner: the object is created for a given class, so that its structure is known for its whole lifetime.

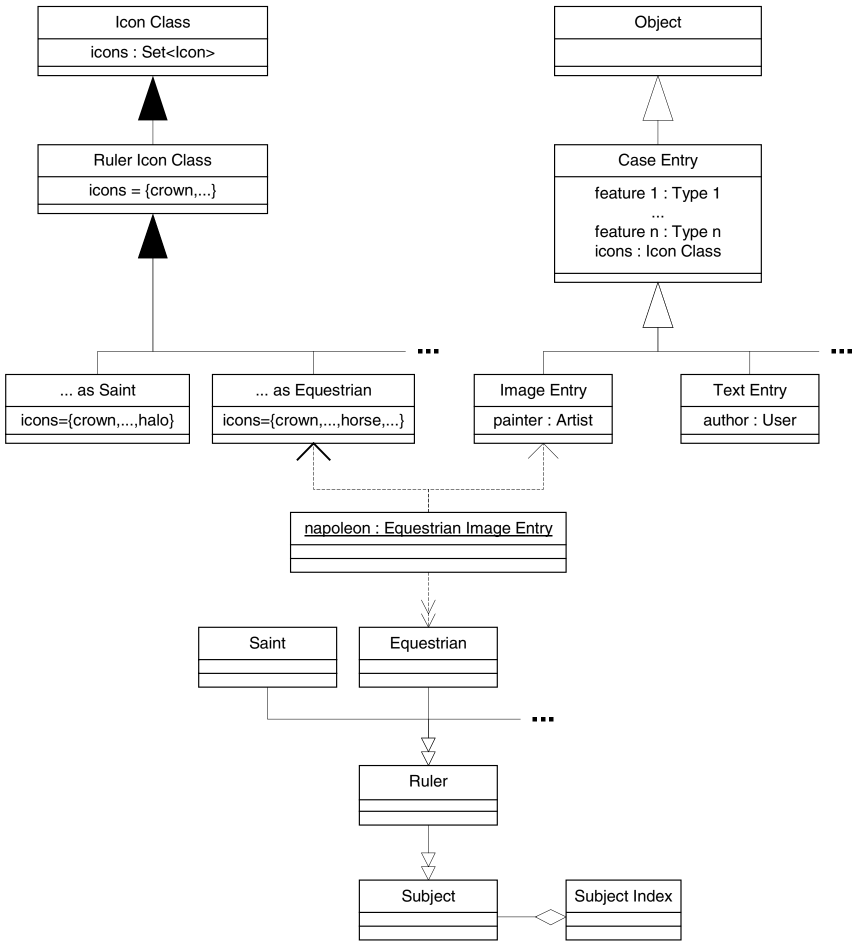


Fig. 10. Subject, object, and icon classes

Important for the SOWing model is the special attribute “icons” which is defined by “Case Entry” and which represents references to an icon class. Icon classes describe sets of icons which relate to subject terms. Icons reflect the content of the described document, e.g. keywords in a text or symbols in an image. As indicated in the upper left of the diagram (Fig. 10), icon classes are ordered in a specialization relationship which is derived from the inclusion of the icon sets. We use the filled-in arrow head to visualize icon class specialization and a thick arrow head for icon class “instantiation”.

Finally, the lower part of Fig. 10 shows a (UML-inspired) formulation of subject classes. We use double-headed arrows for specialization and instantiation. Dual to the (object) class “Object” we introduce a (subject) class “Subject” as the root of the subject class hierarchy. Although subject terms and icon classes are only loosely related, it is very likely that all the entries of the same subject class will have a similar profile over the icons of the corresponding icon class.

The “Subject Index” to which a subject belongs defines the domain to which a subject term contributes. This models the perspectives under which the classification of a description can be viewed. For any given application often only one index will be considered at a time. However, personalization and customization will require the SOWing system to cope internally with several indices simultaneously.

There are two fundamental uses of the structure shown in the diagram, Fig. 10:

- if a subject worker manually establishes the classification relationship between an entry and a subject term (“entry” and “equestrian” in the example of Fig. 10), the entry contributes to the subject term’s definition. For the related icon classes this may mean that the icon class can be derived from or validated by the icons of the entries in an extent;
- vice versa, icons assigned to a description can be matched against an icon class which in return corresponds to a subject term. Using some distance function the SOWing system can derive or propose the classification of an entry.

In this way the subject classification differs substantially from object classification: in contrast to an object class a subject class does not define a uniform structure for all members of its extent. In addition, subject class entries may be members in more than one extent simultaneously and may change subject class membership during their lifetime. In contrast, an entry in its role as object belongs to exactly one object class and this membership is immutable over time.

4.3 Personalization Facilities

We substantiate some of the architectural decisions of the SOWing environment by giving examples from the Warburg Electronic Library. First we look at a the personalization [24] of a description work. Imagine the user downloads a case entry as the following XML document:

```
<picture-card>
  <title>Bonaparte Crossing ...</title>
  <artist>David, Jacques-Louis</artist>
</picture-card>
```

Once a user has copied the description into his personal working environment he is free to modify it at will and might end up with a document like the one shown below.

```
<picture-card>
  <title>Napoleon Crossing ...</title>
  <artist>/artistdb/artist4368.xml</artist>
  <medium>Painting</medium>
</picture-card>
```

Clearly, three changes of a different nature were performed:

- a value change: from “Bonaparte” to “Napoleon”;
- a type change: “artist” now is a reference, no longer a value;
- an additional feature is added: “medium”.

The latter two changes exemplify the semi-structured nature of the descriptions: part of the personalized descriptions conforms to the schema the community chose for descriptions. Another part was added by the user, freely choosing a tag name, the elements content format, and the position where to integrate the new element. This is the kind of liberty the subject workers expect to have in the SOWing environment.

When the community decides to accept and re-integrate the user's contribution into the community's subject index it has to perform the data (eventually also the schema) update [4]. The SOWing server discovers such type changes by tracking the description work [11].

4.4 Case Entry Generation from XML Documents

A SOWing user may submit a work case document in the form of the following XML document:

```
<report>
  In the <medium>Painting</medium>
  <title>Napoleon Crossing ...</title> the artist
  <artist>David, Jacques-Louis</artist> depicts
  Napoleon riding ...
</report>
```

Then the SOWing system will start analyzing the above work document and generate the following initial work case entry:

```
<picture-card>
  <!-- according to the original conceptual model -->
  <title>Napoleon Crossing ...</title>
  <artist>David, Jacques-Louis</artist>
  <!-- additional markup recognized -->
  <medium>Painting</medium>
</picture-card>
```

This automatically generated version of a work case entry serves as the basis for the description worker's SOWing task.

4.5 SOWing Interfaces and External Tool Support

SOWing entities – work cases, documents, case entries, and subjects - may vary widely by their degree of formalization ranging from rather informal entities (to be mediated to humans) to fully formalized structures (to be read and processed by machines). Furthermore, SOWing entities may be consumed and produced by external tools thus requiring a general interfacing technology to the world outside the SOWing platform. A more detailed discussion in the context of XML can be found in Sect. 5.

Our vision of a SOWing system is to *support* a community of users in a long-term process of sharing, evolving and partially formalizing their understanding of a common application domain – and not to *force* them to use a SOWing system. If such a system is to be used it must be possible to create a wide variety of external representations of the entities and relationships maintained by the system.

This is required for two reasons:

- Users want to import the content from such a system into their own working environment (word processors, multimedia authoring tools, expert systems etc.). For this purpose a portable external representation is needed which can be understood by different software systems – at least in part.
- The second reason for a portable and even human readable format is the necessity to facilitate the exchange of the data and the knowledge behind it between different people. Especially those who do not possess an identical or compatible SOWing system will need such an external representation. We consider the extensible markup language as being useful in both scenarios.

On the other hand the SOWing system itself must also be able to process data from various sources. Therefore, either converters between external data and the internal SOWing data model have to be provided or well-defined interfaces to external representations must be available.

5 XML for Subject-Oriented Work

In the previous sections we developed a scenario for a software environment which supports and records subject-oriented work. As soon as the content maintained by such a system is communicated to others or used cooperatively with different systems, the question arises, how to represent content and knowledge about it in a system-independent manner. The Extensible Markup Language and its standards [2], [15] are intended to enhance content structure and coherence and, by doing so, to improve content production and interchange.

Parallel to our work on the SOWing environment much work has been invested in XML-based standards for various content-related services [1]. However, most of these standardization efforts lack a common application scenario which could prove more than the usefulness of isolated standards, e.g., by making two or more of them collaborate. We regard our SOWing system as a cooperative platform for suites of XML-based services.

5.1 XML and Documents

One application of XML in the scenarios mentioned above is its use as a platform-independent notation for content exchange. XML can be read and processed by persons as well as by machines. *Processing* in this context means that machines are able to store XML documents and increase their coherence by performing certain consistency checks, e.g., whether XML data are well-formed or valid according to a certain document type definition. With pure XML, as stated in [10], only syntactical consistency and interoperability can be achieved. If a target machine is supposed to go beyond that point reason about the content of an XML document, domain knowledge has to be hardwired into the processing machine and this knowledge has to be synchronized with assumptions from the content author.

Under this assumption pure XML then could be used to facilitate communication between partners who share a common understanding of the nature of the content being exchanged.

5.2 XML and Semantics

Up to a certain degree XML can be used to represent subject terms and their relationships. However, as [12] argues, important ontological relationships (e.g., "subclass-of" or "instance-of") can not be modeled directly by a XML document type definition. Another problem arises because XML allows different ways to model the same relationship and no support for some notion of equivalence. Properties of a concept can be modeled in at least two ways: as an XML element on its own or as an attribute of another element. Supposing a document type description is used for some ontological statement, the receiver of a document based on this statement has to "know" that this document type represents an ontology and how it can be derived from the document type definition. Currently several proposals for a common formalism for ontology representation are being discussed [10], [12].

The Resource Description Framework (RDF, see, for example, [15]) provides primitives which facilitate the representation of ontologies in a much more natural way compared to (pure) XML. RDF can be implemented on top of XML and is suggested to be used as a basic framework for the definition of common ontology interchange languages, for example, OIL [7].

We expect that by working within the SOWing environment in combination with the above standards we can substantially improve the re-use of publication work and thus simplify the overall publication process. Furthermore, the understanding of documents for readers outside the community for which a work was originally published can be improved and machine reasoning about the works becomes feasible to the extent to which as both, the SOWing system and the processing machine, share subject index information.

5.3 XML and Activities

As mentioned earlier our notion of "work" refers not only to the result of a product process, i.e., to documents on certain concepts or ideas, but to the entire process by which the document is created. Through its reflective capabilities to observe the production process the SOWing system can contribute to questions such as

- in which sequence were the production steps carried out ?
- how long did each production step take ?
- what are the influences leading to a concrete object production sequence ?

For works belonging to the past this information usually is not available or has to be derived from other sources. For description work as carried out by the Art Historians in the WEL project, however, information about the production process of such a description work can be collected almost automatically by the SOWing environment. Work cases being "first class citizens" can themselves become subjects to later research and one does not need to "guess" what the circumstances of a production may have been but one *knows* what these circumstances were – at least to the extent the SOWing environments model documents them.

In this sense the SOWing environment can document the process which led to a certain document and enhance the understanding of the "work" behind the document. On the other hand this process description can be used to derive recommendations for future production processes in the sense of instructions that have to be carried out for tasks similar to a successfully completed process [14]. Well-documented production processes can serve as templates for future production and contribute to process standardization [25]. XML-based languages such as XRL (eXchangeable Routing Language [31]) can be used to represent such processes.

In summary, we expect a twofold contribution when using XML as a common SOWing interface: XML-based tools will substantially and rapidly enhance the SOWing functionality and, in reverse, the SOWing model will give a semantic underpinning and connectivity to XML services and thus significantly improve their usability.

6 Summary and Outlook

Our SOWing platform, experiments and the SOWing project as a whole aim at relating, organizing and defining subjects and documents as well as the work behind it. In an interdisciplinary project with our Art History colleagues who are working in the subject area of "Political Iconography" we gained substantial insight into their *Subject-Oriented Working* (SOWing) needs and into initial requirements for a generic SOWing platform. In this paper we outlined the project, its basic models, their generalization as well as our initial experiences with prototypical SOWing implementations and compared our work with various XML-related activities.

On the modeling level we improved our understanding of
- the basic SOWing entities and their relationships;
- the notion of work, i.e., the production context of content;
- the role of key icons and key words for content-based subject definition.

On the system level SOWing project is currently investigating the requirements for
- a generator-based architecture for SOWing entities and relationships;
- reflective system technology and its use for advanced SOWing services;
- customized and personalized SOWing indices;
- XML-based tool interoperability.

Future large scale content-oriented project work will have to interact with a substantial number of SOWing indices and, therefore, requires a technology for "plugable SOWing arrays". SOWing indices have to deliver their content through a wide variety of document types ranging from media documents laid out for human interaction to structured (and typed) documents for machine consumption. Finally, SOWing support for the modeling, management and enactment of content-oriented work has to be further improved.

Acknowledgement. We would like to thank the Ministry for Science and Research, County of Hamburg, and the Warburg Stiftung, Hamburg, for their continuous support of our research. In addition we gratefully acknowledge a research grant from the Deutsche Forschungsgemeinschaft, DFG, in support of our R&D project on "Cooperative Reference Libraries".

References

1. Berners-Lee, Tim, Fischetti, Mark: Weaving the Web; the original design and ultimate destiny of the World Wide Web by its inventor. Harper, San Francisco (2000)
2. Bray, Tim, Paoli, Jean, Sperberg-McQueen, C. M., Maler, Eve: Extensible Markup Language (XML) 1.0 (2nd Edition). http://www.w3.org/TR/2000/REC-xml-20001006 (2000)
3. Bruhn, M.: The Warburg Electronic Library in Hamburg: A Digital Index of Political Iconography. In: Visual Resources, Vol XV (1999) 405-423
4. Buckingham Shum, Simon: Negotiating the Construction and Reconstruction of Organizational Memories. Journal of Universal Computer Science, vol. 3, no. 8 (1997) 899-928
5. CoreMedia AG: Homepage. http://www.coremedia.com (2001)
6. Coulter, Neal, French, James, Glinert, Ephraim, Horton, Thomas, Mead, Nancy, Rada, Roy, Ralston, Craig, Rodkin, Anthony, Rous, Bernard, Tucker, Allen, Wegner, Peter, Weiss, Eric, Wierzbicki, Carol: ACM Computing Classification System 1998: Current Status and Future Maintenance. Technical report, http://www.acm.org/class/1998/ccsup.pdf (1998)
7. Cover, R.: Ontology Interchange Language. http://xml.coverpages.org/oil.html (2001)
8. De Michelis, Giorgio, Dubois, Eric, Jarke, Matthias, Matthes, Florian, Mylopoulos, John, Schmidt, Joachim W., Woo, Carson, Yu, Eric: A Three-Faceted View of Information Systems. In: Communications of the ACM, 41(12) (1998) 64-70
9. Deacon, Terrence W.: The Symbolic Species. The co-evolution of language and the brain. W. W. Norton & Company, New York/London (1997)
10. Decker, S., van Harmelen, F., Broekstra, J., Erdmann, M., Fensel, D., Horrocks, I., Klein, M., Melnik, S.: The semantic web: The roles of xml and rdf. In: IEEE Expert, 15(3) (2000)
11. Dourish, P., Bellotti, V.: Awareness and Coordination in Shared Workspaces. In: Proceedings of ACM CSCW'92 Conference on Computer-supported Cooperative Work, ACM-Press (1992) 107-114
12. Fensel, D.: Relating Ontology Languages and Web Standards. In: Informatik und Wirtschaftsinformatik. Modellierung 2000, Foelbach Verlag (2000)
13. Gruber, T. R.: A translation approach to portable ontology specifications. Technical Report KSL 92-71, Computer Science Department, Stanford University, California (1993)
14. Khoshafian, Setrag, Buckiewicz, Marek: Introduction to Groupware, Workflow, and Workgroup Computing. John Wiley & Sons, Inc., New York (1995)
15. Lassila, O., Swick, R. R.: Resource Description Framework (RDF) Model and Syntax Specification. Recommendation. W3C. http://www.w3.org/TR/REC-rdf-syntax/ (2000)
16. Matthes, F., Schröder, G., Schmidt, J.W.: Tycoon: A Scalable and Interoperable Persistent System Environment. In: Atkinson, Malcom P., Welland, Ray (eds.): Fully Integrated Data Environments, ESPRIT Basic Research Series, Springer-Verlag (2000) 365-381
17. Maurer, Hermann, Lennon, Jennifer: Digital Libraries as Learning and Teaching Support. In: Journal of Universal Computer Science, vol. 1, no. 11 (1995) 719-727
18. McLuhan, Marshall: Understanding Media. The Extensions of Man. The MIT Press, Cambridge/Massachusetts/London (1964, 1994)

19. Niederée, C., Hattendorf, C., Müßig, S. (with J.W. Schmidt und M. Warnke): Warburg Electronic Library – Eine digitale Bibliothek für die Politische Ikonographie In: uni-hh Forschung, Beiträge aus der Universität Hamburg, XXXI (1997) 6-16.
20. Nürnberg, Peter J., Schneider, Erich R., Leggett, John J.: Designing Digital Libraries for the Hyperliterate Age. In: Journal of Universal Computer Science, 2 (9) (1996) 610-622
21. Raulf, M., Müller, R., Matthes, F., Scheunert, K.J., Schmidt, J.W.: Subject-oriented Document Administration for Internet-based Project Management (in German). In: Proceedings "Management and Controlling of IT-Projects", dpunkt.verlag, Heidelberg (2001)
22. Rostek, Lothar, Möhr, Wiebke, Fischer, Dietrich: Weaving a Web: The Structure and Creation of an Object Network Representing an Electronic Reference Work. In: Fankhauser, P., Ockenfeld, M. (eds.): Integrated Publication and Information Systems. 10 Years of Research and Development at GMD-IPSI, Sankt Augustin: GMD (1993) 189-199
23. Schmidt, J. W., Matthes, F.: The DBPL Project: Advances in Modular Database Programming. Information Systems, 19(2) (1994) 121-140
24. Schmidt, J.W., Schröder, G., Niederée, C., Matthes, F.: Linguistic and Architectural Requirements for Personalized Digital Libraries. In: International Journal on Digital Libraries, 1(1) (1997)
25. Schmidt, Joachim W., Sehring, Hans-Werner: Dockets: A Model for Adding Value to Content. In: Akoka, Jacky, Bouzeghoub, Mokrane, Comyn-Wattiau, Isabelle, Métais, Elisabeth (eds): Proceedings of the 18th International Conference on Conceptual Modeling, volume 1728 of Lecture Notes in Computer Science, Springer-Verlag (1999) 248-262
26. Schmidt, Joachim W., Sehring, Hans-Werner, Warnke, Martin: The Index for Political Iconography and the Warburg Electronic Library (in German). In: Proceedings of the International Symposium on "Archiving Processes", Köln (2001)
27. Simone, Carla, Divitini, Monica: Ariadne: Supporting Coordination through a Flexible Use of the Knowledge on Work Processes. In: Journal of Universal Computer Science, vol. 3, no. 8 (1997) 865-898
28. Smith, John Miles, Smith, Diane C. P.: Database Abstractions: Aggregation and Generalization. In: TODS 2(2) (1977) 105-133
29. Sowa, John F.: Knowledge Representation, Logical, Philosophical, and Computational Foundations. Brooks/Cole, Thomson Learning (2000)
30. Svenonius, Elaine: The Intellectual Foundation of Information Organization. The MIT Press, Cambridge/Massachusetts/London, England (2000)
31. van der Aalst, W. M. P., Kumar, A.: XML Based Schema Definition for Support of Interorganizational Workflow. http://tmitwww.tm.tue.nl/staff/wvdaalst/Workflow/xrl/isr01-5.pdf
32. van Waal, Henri: ICONCLASS – An iconographic classification system. North-Holland Publishing Company, Amsterdam/Oxford/New York, completed and edited by L.D. Couprie, R.H. Fuchs, E. Tholen, G. Vellekoop, a.o. (1973-85)
33. Warnke, Martin: Bildindex zur politischen Ikonographie. Forschungsstelle Politische Ikonographie, Kunstgeschichtliches Seminar der Universität Hamburg (1996)
34. Warnke, Martin (ed.): Der Bilderatlas Mnemosyne. Unter Mitarbeit von Claudia Brink. Akademie Verlag, Berlin (2000)
35. Webster's Third New International Dictionary of the English Language, Chicago (1996)

Query Optimization through Removing Dead Subqueries

Jacek Płodzień[1] and Kazimierz Subieta[2,1]

[1] Institute of Computer Science PAS, Warsaw, Poland
{jpl,subieta}@ipipan.waw.pl
[2] Polish-Japanese Institute of Information Technology, Warsaw, Poland

Abstract. A dead subquery is a part of a query not contributing to the final query result. Dead subqueries appear mostly due to querying views. A method of detecting and eliminating dead subqueries is presented. It assumes that views are processed by query modification, which macro-substitutes a view invocation with the corresponding view definition. The method is founded on a new semantic framework of object-oriented query languages, referred to as the stack-based approach. Dead parts are detected through static (compile-time) analysis of scoping and binding properties for names occurring in a query. The method is explained by a pseudo-code algorithm and illustrated by examples.

1 Introduction

A disadvantage of using *virtual* (non-materialized) database views is that a view often calculates more than it is necessary in a particular view invocation. For example, a view calculates, among others, average salaries in departments, but in a particular query invoking the view these values are not used. Hence the straightforward approach to view processing may result in a significant waste of processing time. The problem has been partly solved for relational query languages through the technique known as *query modification* [8]: a view definition is treated as a macro-definition, where each view invocation is textually substituted with the query from the body of the view definition. Then, the parts of the view definition not used in a particular query are removed from the resulting query. Such parts are called *dead subqueries* (*parts*), because they do not contribute to the final result. (Dead subqueries may also appear in queries without views; however, this is rather rare.)

Query modification has been adopted and generalized for object-oriented query languages such as OQL [12]. This technique prepares a foundation for our method of detecting and removing dead subqueries. Such optimization is performed through rewriting a query into a semantically equivalent query having no dead subqueries. After removing dead subqueries, the resulting query can be further processed by other optimization methods, e.g. based on factoring out independent subqueries, changing the order of operators, indices [4,5,6].

In a general case of object-oriented query languages and views, the problem of removing dead subqueries is not easy. A query resulting from query modification can be complex, nested, can involve inheritance and encapsulation, and can invoke methods or virtual attributes. It can also contain definitions of auxiliary names (i.e., correlation variables, variables bound by quantifiers, etc) and can be built upon various operators such as selection, dot (i.e., projection/navigation), dependent join, quantifiers, arithmetic functions, aggregate functions, and others. Moreover, views can be nested; i.e. a particular view can invoke other views.

As far as we have investigated the literature, there is no other paper considering the problem of removing dead subqueries for object-oriented query languages and views. Probably this is caused by the fact that object views are not developed to a satisfactory degree, despite many papers (for a discussion see [9]).

Nonetheless, a related idea is discussed for the relational approach. For instance, in [3] the authors consider a case when redundant joins of base relations appear as a result of using views defined (directly or indirectly) against those relations. A similarity between the method discussed in [3] and our solution is that the general idea of both techniques is to avoid performing unnecessary operations. The main difference is that [3] deals with joins, which are the most expensive operations in relational databases, and is restricted to CPS-queries, while our method concerns any (arbitrarily complex) subqueries whose results are not used. Joins are not the case in the object approach, because foreign keys are usually replaced with links. Our approach has potential to cover cases when dead subqueries are not connected by joins.

Our (fairly general) approach assumes that object views are first-class stored functional procedures [9]. To deal with scoping and binding rules for names occurring in queries/programs, we follow the *Stack-Based Approach* (SBA) to object-oriented query languages [5,10,11]. The approach assumes that some query operators (**where**, dot, dependent/navigational join, quantifiers, etc) act on an environment stack similarly to program blocks or procedures, that is, they open a new section for binding names.

In our research we focus on static (i.e., compile-time) query optimization. In order to gather all the information that is needed to determine whether a query can be optimized, a special phase of query processing – so-called *static analysis* – is performed [7]. During static analysis we simulate the behavior of the run-time *query result stack* (QRES) and the *environment stack* (ES) through the corresponding static stacks (i.e., compile-time versions of ES and QRES). They keep signatures of values returned by queries and signatures of run-time environment stack sections and subsections. Recursive analysis of these signatures and their relationships with the syntax tree of the processed query makes it possible to detect unused environment stack subsections, then to navigate from these subsections to corresponding parts of the query syntax tree. These parts are cut off; i.e. dead subqueries are removed.

The general query processing architecture is presented in Fig. 1. First, the text of a query is parsed and its syntax tree is constructed. Then, the query is optimized by rewriting. Static analysis (performed by a query optimizer) involves a metabase (a special data structure obtained from a database schema), a structure simulating the environment stack (static ES denoted by S_ES), and a structure simulating the query result stack (static QRES denoted by S_QRES).

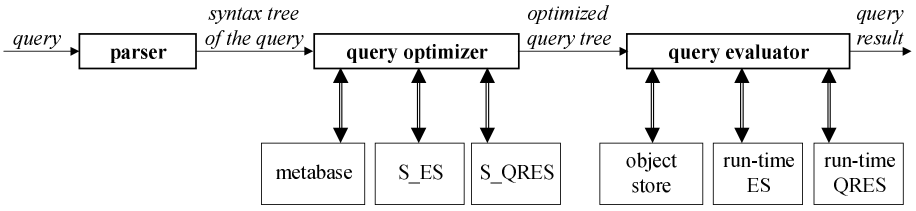


Fig. 1. General architecture of query processing

After optimization the query is evaluated; the evaluation involves run time structures, that is, an object store, ES and QRES.

An important aspect of query optimization is a cost model. However, because removing dead subqueries *always* improves performance, it should be applied whenever possible. As a consequence, there is no need to assess performance improvement during optimization, even for cost-based optimization. Therefore, we do not consider a cost model in this paper.

The queries in the paper are defined for an example database whose schema (the class diagram in a little modified UML) is shown in Fig. 2. The classes *Lecture*, *Student*, *Professor* and *Faculty* model lectures attended by students and given by professors working in faculties, respectively. *Professor* objects can contain multiple complex *prev_job* subobjects (previous jobs). The name of a class, attribute, etc, is followed by its cardinality, unless it is 1. All the properties are public.

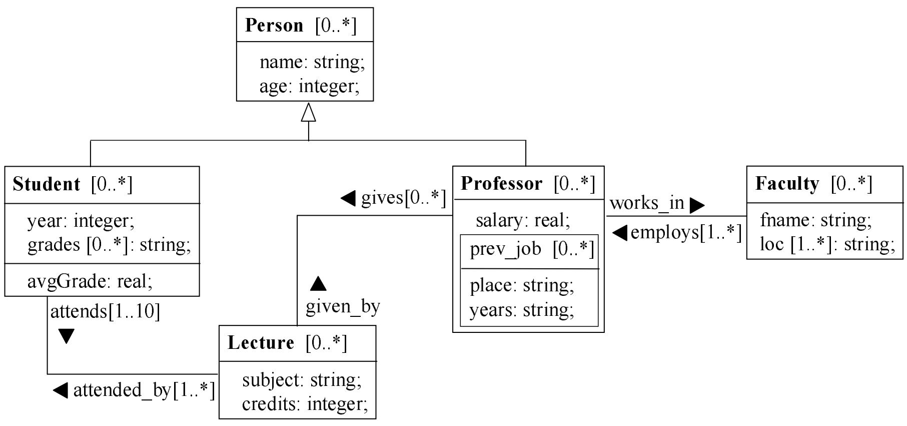


Fig. 2. The class diagram of the example database

The rest of the paper is organized as follows. Section 2 presents a few SBA concepts important for the paper. Section 3 describes general properties of dead subqueries. In Sect. 4 we discuss in detail our method of finding and removing a dead subquery through rewriting. We also present an example of applying the method to a query. In Sect. 5 we discuss how to take into account all dead subqueries in a query. In Sect. 6 we extend our discussion by considering other cases when dead parts can appear. Section 7 concludes our discussion.

2 Essential Concepts

SBA is an attempt to build a uniform semantic foundation for integrated query and programming languages. The approach is abstract and universal, which makes it relevant to a general object model being an abstraction over e.g. ODMG OQL [2] and XML models. SBA has its own query language - SBQL, which is a syntactic variant of OQL with a fully formalized semantics. In this section we explain a few SBA concepts essential for our method of removing dead subqueries. However, we assume that the reader is at least a little familiar with the general idea of SBA.

2.1 Subsections

Normally, an ES section induced by a particular non-algebraic query operator consists of all binders that are returned by the *nested* function for the currently being processed row of a result table. Thus we cannot distinguish between binders created for the results of different subqueries. To make such a distinction possible we introduce the concept of *subsection*. It is a part of an ES section, which contains binders constructed for one element of the row currently being processed for which a non-algebraic operator has opened that section.

Subsections make it possible to assign each binder of a given section to the appropriate subquery of a query being evaluated (represented as a syntax tree). Each subsection contributes to partial results of further subqueries, which in turn generate other subsections, etc., up to achieving the final result. Through assigning each subsection the node of the corresponding syntax tree of the query, we can propagate identifiers of tree nodes in order to obtain information which subsections contribute to the result of an outer query. This process is performed recursively. Because subsections are associated 1:1 with subqueries (i.e., the corresponding syntax subtrees), in this manner we detect dead subqueries (i.e., subqueries associated with unused subsections on ES).

This process is performed on S_ES and S_QRES. During static analysis we deal with *section signatures* and *table signatures*, which model run-time ES sections and query results, respectively. To detect dead subqueries we use *subsection signatures*, which model run-time subsections. This part of static analysis is discussed in detail in Sect. 4.

2.2 Views and Query Modification

Views[1] raise the level of abstraction and modularity while building an application, but may result in poor performance due to overhead connected with evaluation of a view during each invocation. To deal with this problem, in relational databases queries invoking views are optimized through *query modification.* It combines a query containing view invocations with the definitions of the views being invoked. The resulting query has no references to views and can be optimized at a textual level by rewriting rules. The advantage of this technique is twofold. First, it makes it possible to avoid performance overhead related to processing views. Second, it enables to use other optimization methods, e.g. indices. For these qualities we have adopted the query modification technique to object-oriented queries [12].

In SBA views are treated as first-class (i.e., stored in a database) functional procedures that can return a collection [11]. They can be defined by a single query or by a sequence of statements accomplishing a complex algorithm, possibly with a local environment. Moreover, they enable the programmer to compose a virtual object from several stored objects. Such views can be recursive, can have parameters, and can be updateable. The idea of views in SBA supports a fairly general object model including classes, methods, inheritance, etc.

Assuming some discipline concerning the functionality, syntax and semantics of view definitions, the query modification is reduced to regular *macro-substitution.* Such an approach to views is assumed in OQL [2] through the "define" clause. Query languages and views for XML databases are currently the subject of extensive research, but there is little agreement concerning this issue, see e.g. [1].

In this section we discuss view processing in the context of query optimization based on rewriting. In SBQL views are defined as follows (we omit typing):

```
view Freshmen
begin
    return Student where year = 1
end;
```

Functions, like the one above, can be invoked in queries; an example ("get names and ages of freshmen older than 30") is presented below:

(*Freshmen* **where** *age* > 30).(*name*, *age*)

This query is evaluated as follows. First, the name *Freshmen* is bound in the ES base section; then the function is invoked. It has no parameters and its local environment is empty, so the ES section for this invocation will contain only the return address. The body of *Freshmen* consists of a single query which is evaluated in the standard way. The result of this query consists of references to appropriate *Student* objects. Then, the function is terminated and the ES section implied by that function is popped. Next, the result returned by *Freshmen* is processed by the **where** operator: the result contains references to those *Student*

[1] We deal only with *virtual* views. Materialized views present another subject, not relevant to this paper.

objects, for which the value of the *age* attribute is greater than forty. Then, this result is processed by the dot. The final result consists of pairs of references to the *name* and *age* subobjects for the selected *Student* objects.

The functions, introduced as views, possess the following properties:

- They do not introduce their own local environments.
- They have no parameters.
- They are dynamically bound.
- The only statement of their bodies is a single query.

Functions having these properties are semantically equivalent to *macro-definitions*. We have constructed the syntax and semantics of SBQL in such a way that it is fully orthogonal with respect to combination of any queries, including queries containing auxiliary names (e.g., names defined by the **as** operator, variables bound by quantifiers, new names used in view definitions, etc.). For example, after query modification the above query will have the form:

((*Student* **where** *year* = 1) **where** *age* > 30).(*name*, *age*)

Thus (in contrast to [8]) our query modification method is extremely simple: we just textually *macro-substitute* a view invocation with the body of this view. As will be shown in further examples, new names introduced in the view definition for attributes of virtual objects present for us no irregularity or difficulty. For a more detailed discussion of the issue see [12].

3 Properties of Dead Subqueries

While searching for dead subqueries we have to exclude subqueries that create *directly* or *indirectly* the result of a query. For instance, in the query

Professor ⋈ (*works_in.Faculty*)

the subqueries *Professor* and *works_in.Faculty* are not dead because their results are *directly* parts of the result of the whole query. Similarly, in the query

Professor.name

the subquery *Professor* is not dead because it *indirectly* contributes to the final result. Subqueries that do not satisfy either of those two conditions are dead and can be removed. For example, in the query

(*Professor* ⋈ (*works_in.Faculty*)).*age*

the subquery *works_in.Faculty* does not contribute to the result of the query either directly or indirectly and thus is dead.

In some (rare) cases subqueries that are apparently dead cannot be removed. Due to space limit we do not discuss this degenerated case in the paper.

Before discussing our method for detecting dead subqueries we present a few examples and discuss the general idea.

3.1 Examples of Dead Subqueries

The most frequent case when a query has dead subqueries is when it contains a view invocation and performs navigation starting from the result returned by the view. Let us show this by an example. The view below gets for each professor his/her name, salary, and calculates a bonus he/she should receive, giving them auxiliary names *n*, *s*, and *b*, respectively. The view can be considered the definition of virtual objects named *ProfNameSalaryBonus* having attributes *n*, *s*, and *b*.

```
view ProfNameSalaryBonus
begin
   return Professor.
      (name as n, salary as s, ((0.5*salary + 10*age)*count(gives)) as b)
end;
```

To retrieve through this view the names of all professors stored in a database, we have to ask the following query:

ProfNameSalaryBonus.n

After macro-substitution it has the form:

(*Professor.*
   (*name* **as** *n*, *salary* **as** *s*, ((0.5**salary* + 10**age*)*count(*gives*)) **as** *b*)).*n*

Note that the final projection (i.e. ".*n*"), which creates the final result of the whole query, refers to the result of the subquery *name* **as** *n* but does not refer to the results of the subqueries *salary* **as** *s* and ((0.5**salary* + 10**age*)*count(*gives*)) **as** *b*. Hence the latter two subqueries are dead and can be removed. After the transformation the query has the form (*Professor.*(*name* **as** *n*)).*n* which can be further rewritten to *Professor.name* by another optimization method (removing unnecessary auxiliary names [4]).

Another non-algebraic operator that may cause a subquery to be dead is a quantifier. The analogy with the case for the dot operator is especially well seen if a query is written in the form reflecting the schema of its evaluation, that is, if quantifiers are used as infix operators [4].

3.2 Summary

As we have discussed, a query may have dead subqueries if it contains dot operators or quantifiers. The reason is that a dot and quantifiers have a special property: it is possible for them to consume only a part of the result of their left subquery, which makes the rest of that left operand not contribute to the result of the entire query.

Other non-algebraic operators considered in SBA, like a selection or a dependent join, do not have this property. For example, $\bowtie$ consumes the whole result of its left subquery thus does not cause any subquery to be dead. The **where** operator can use only a part of the result of its left operand, but the whole of that left subquery determines the result of a query. This means that the selection operator does not cause any subquery to be dead either (with some exceptions; see Sect. 6). We illustrate this by the following query ("get professors and lectures provided by them assuming that professors are older than fifty"):

(*Professor* $\bowtie$ (*gives.Lecture*)) **where** *age* > 50

Obviously, it is impossible to find a query after **where** which will cause that some subquery before **where** will be dead. A similar property holds for $\bowtie$.

4 Detecting Dead Subqueries

In order to optimize a query with regard to dead parts, a modified static analysis is required. In comparison to [7], a little modified versions of the *static_eval* procedure and the *static_nested* function are used: *ds_static_eval* and *ds_static_nested*, respectively. Moreover, the concept of signatures must be modified to keep the following additional information:

- In a modified table signature (on S_QRES), each element is augmented with a pointer to the root of the appropriate syntax query subtree (representing a subquery which returns the result modeled by this element).
- Similarly, in a modified section signature (on S_ES), each subsection signature is augmented with a pointer to the root of the appropriate syntax query subtree. Bottom sections' signatures (for which subdivision into subsections is irrelevant) are augmented with NULL pointers.

Additionally, while binding a name in some subsection signature, the subquery in the syntax tree, for the result of which that signature was built, is flagged as "*REFERRED_TO*". This flag means that this particular query syntax subtree has been used to evaluate its outer query (hence the corresponding subquery is not dead). A part of the *ds_static_eval* procedure is presented below; it covers only the case of non-algebraic operators after both q_1 and q_2 were evaluated but before the final result is created. For details and other cases, e.g., algebraic operators, see [4].

Before applying *ds_static_eval* to a given query we can scan its definition to check whether it can potentially have dead parts, that is, if it contains dots and/or quantifiers. If it does not, then we know that it has no dead parts; we do not have to use the *ds_static_eval* procedure to optimize it.

The auxiliary function *Subquery*(*ss*) returns the pointer to the subquery in the syntax tree for the result of which (sub)section signature *ss* was built; for a bottom section signature it returns NULL.

```
if ($\theta \in$ {dot, $\forall$, $\exists$}) then
  for each subsection signature ss in new_scope do
    (*check if that (sub)result of $q_1$ is used*)
    if (the subquery pointed to by Subquery(ss) is not
        REFERRED_TO) then (*it's dead*)
    begin
      if (the subquery pointed to by Subquery(ss) is
          the right operand of a dot operator) then
        (*remove that dead part and a ''future'' dead part*)
        remove that dot and its both subqueries;
      else
        (*remove only the dead part*)
        remove the subquery pointed to by Subquery(ss);
      remove the operator connecting the eliminated
      part to the rest of the whole query;
      remove from q1result the element for which ss was
      constructed;
    end;
```

Note that because in queries of the form $q_1\ \theta\ q_2$ (where θ is a dot or quantifier) only subquery q_2 can make use of the result of q_1 when evaluation of $q_1\ \theta\ q_2$ ends, the result of q_1 cannot be used by any subquery anymore. Therefore, right after that evaluation we check whether the result of q_1 was used: we do it by checking in the syntax tree which parts of the scope opened for the result of q_1 were referred to. Those that were not referred to will not be referred to afterward, thus the corresponding subqueries are dead and can be removed.

In the case when a dead part is the right operand of a dot operator, we remove not only that dead part, but also the left operand of that dot operator, since after removing its right operand its left operand would become dead. Additionally, we remove from the S_QRES table signature those elements that are the results of those dead parts, because we do not need to analyze them any longer.

Note that since the algorithm is performed at compile time, its cost does not influence the cost of run-time query processing.

Example

We will explain our method by a detailed example. The view

view *LectProfManyCredits*
begin
  return (*Lecture* ⋈ *given_by.Professor*) **where** *credits* > 5
end;

returns lectures whose credits are greater than five together with the professors giving them. To get only the subjects of those lectures via this view, we can ask the query

LectProfManyCredits.subject

After macro-substitution it has the form

((*Lecture* ⋈ *given_by.Professor*) **where** *credits* > 5).*subject* (i)

Its syntax tree is shown in Fig. 3 (we omit non-terminal grammar symbol nodes that appear in this tree).

Query (i) contains dot operators, so it may have dead parts. The states of the static stacks during a modified static analysis of the query are presented in figures from Fig. 4 to Fig. 7. For each name being bound we show a pair denoting the stack size and the binding level, and for each non-algebraic operator being evaluated we show the number of the section it opens. If the section signature in which a name is bound consists of more than one subsection signature, we label each subsection signature by augmenting the number of the section signature with *a*, *b*, *c*, etc, where *a* denotes the leftmost one. Pointers to subqueries in the syntax tree are designated by invocations of the *Root*(*query*) function, which returns the pointer to the root of (sub)query *query* in its syntax tree.

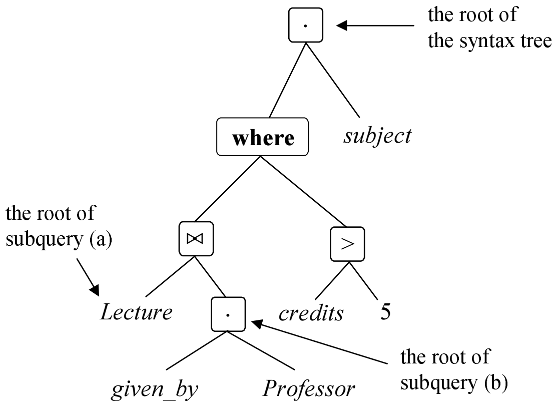


Fig. 3. The syntax tree for query (i)

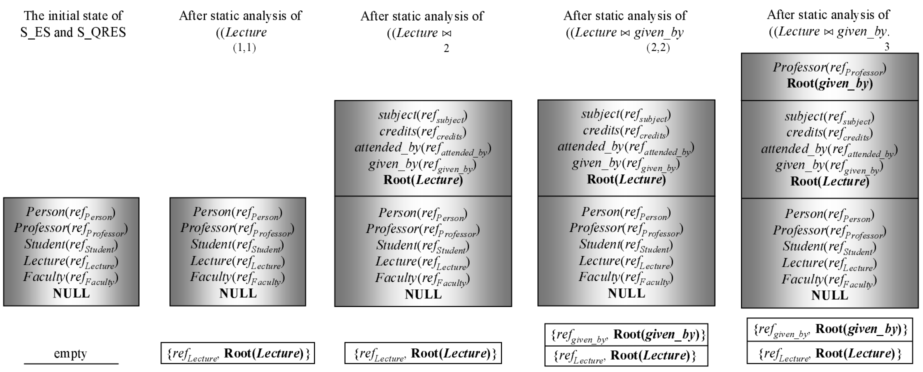

The initial state of S_ES and S_QRES
Person(ref_Person) Professor(ref_Professor) Student(ref_Student) Lecture(ref_Lecture) Faculty(ref_Faculty) NULL
empty
After static analysis of ((Lecture (1,1)
{ref_Lecture, Root(Lecture)}
After static analysis of ((Lecture ⋈ 2
subject(ref_subject) credits(ref_credits) attended_by(ref_attended_by) given_by(ref_given_by) Root(Lecture)
After static analysis of ((Lecture ⋈ given_by (2,2)
{ref_given_by, Root(given_by)}
After static analysis of ((Lecture ⋈ given_by. 3
Professor(ref_Professor) Root(given_by)

Fig. 4. The first part of the analysis

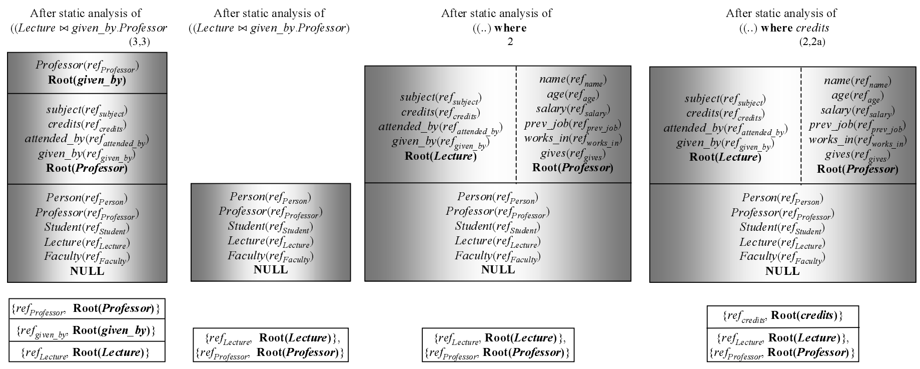

After static analysis of ((Lecture ⋈ given_by.Professor (3,3)
Professor(ref_Professor) Root(given_by)
subject(ref_subject) credits(ref_credits) attended_by(ref_attended_by) given_by(ref_given_by) Root(Professor)
Person(ref_Person) Professor(ref_Professor) Student(ref_Student) Lecture(ref_Lecture) Faculty(ref_Faculty) NULL
{ref_Professor, Root(Professor)}
{ref_given_by, Root(given_by)}
{ref_Lecture, Root(Lecture)}
After static analysis of ((Lecture ⋈ given_by.Professor)
{ref_Lecture, Root(Lecture)}, {ref_Professor, Root(Professor)}
After static analysis of ((..) where 2
subject(ref_subject) credits(ref_credits) attended_by(ref_attended_by) given_by(ref_given_by) Root(Lecture)
name(ref_name) age(ref_age) salary(ref_salary) prev_job(ref_prev_job) works_in(ref_works_in) gives(ref_gives) Root(Professor)
After static analysis of ((..) where credits (2,2a)
{ref_credits, Root(credits)}

Fig. 5. The second part of the analysis

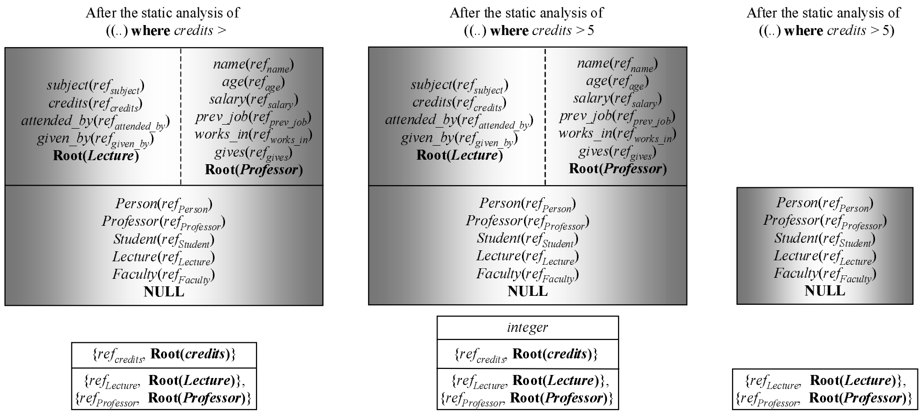

After the static analysis of ((..) where credits >
After the static analysis of ((..) where credits > 5
After the static analysis of ((..) where credits > 5)
subject(ref_subject) credits(ref_credits) attended_by(ref_attended_by) given_by(ref_given_by) Root(Lecture)
name(ref_name) age(ref_age) salary(ref_salary) prev_job(ref_prev_job) works_in(ref_works_in) gives(ref_gives) Root(Professor)
Person(ref_Person) Professor(ref_Professor) Student(ref_Student) Lecture(ref_Lecture) Faculty(ref_Faculty) NULL
integer
{ref_credits, Root(credits)}
{ref_Lecture, Root(Lecture)}, {ref_Professor, Root(Professor)}

Fig. 6. The third part of the analysis

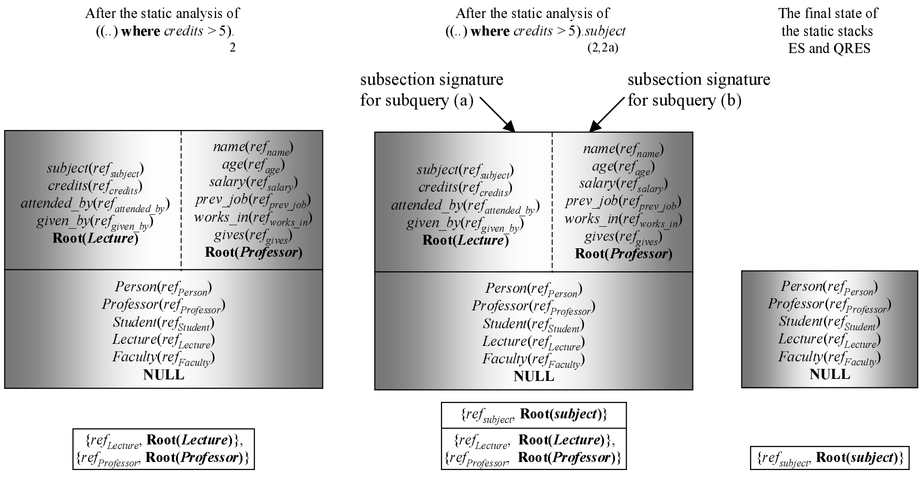

After the static analysis of ((..) where credits > 5). 2
After the static analysis of ((..) where credits > 5).subject (2,2a)
The final state of the static stacks ES and QRES
subsection signature for subquery (a)
subsection signature for subquery (b)
subject(ref_subject) credits(ref_credits) attended_by(ref_attended_by) given_by(ref_given_by) Root(Lecture)
name(ref_name) age(ref_age) salary(ref_salary) prev_job(ref_prev_job) works_in(ref_works_in) gives(ref_gives) Root(Professor)
Person(ref_Person) Professor(ref_Professor) Student(ref_Student) Lecture(ref_Lecture) Faculty(ref_Faculty) NULL
{ref_subject, Root(subject)}
{ref_Lecture, Root(Lecture)}, {ref_Professor, Root(Professor)}

Fig. 7. The fourth part of the analysis

In Figs. 5, 6, and 7 we present the section signature (at the top of S_ES) containing subsection signatures for subqueries (a) and (b). No bindings through the entire analysis in the right subsection corresponding to the subquery (b) means that the subquery *given_by.Professor* is dead.

We optimize (i) by pruning its syntax tree: we cut off the subtree modeling the dead part (along with $\bowtie$). The new, optimized form of the syntax tree is shown in Fig. 8. It models the query (*Lecture* **where** *credits* > 5)*.subject.*

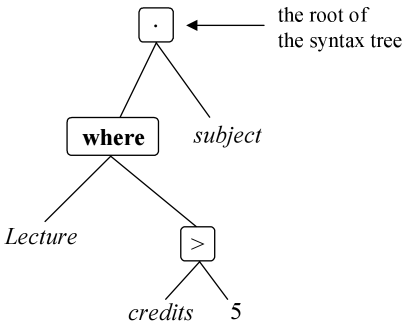


Fig. 8. The form of the syntax tree for query (i) without the dead subquery

5 How to Rid a Query of All of Its Dead Parts

The method presented in the previous section makes it possible to find dead parts of a query and remove them. As we will discuss in this section, removing dead parts may cause other subqueries to become dead. That is, in some cases some parts of a query may turn out to be dead after other dead parts of that query have been removed. Let's start with an example. Suppose that we have a view, which returns lectures along with professors who give them, and faculties in which those professors work:

view *LectProfFaculty*
begin
  return *Lecture* ⋈ *given_by.Professor* ⋈ *works_in.Faculty*
end;

If we want to use this view to get the subjects of all lectures, we can ask the query

LectProfFaculty.subject

After macro-substitution it has the form:

(*Lecture* ⋈ *given_by.Professor* ⋈ *works_in.Faculty*).*subject*

The only dead part is *works_in.Faculty* (it's the biggest dead part). The subquery *given_by.Professor* is not used by the subquery *subject*, but is not dead since it is used by the dead subquery; thus it cannot be eliminated. After removing the dead part the query has the form

(*Lecture* ⋈ *given_by.Professor*).*subject*

Note that now the part *given_by.Professor* is dead. The reason is that the subquery, which used its result, has been removed. Therefore the method should be performed in a loop: in one iteration one modified static analysis of the current form of a query would be rendered, during which all dead parts would be detected and removed; the loop would end when no dead part were found.

6 Other Cases

So far we have considered queries for which a dead subquery was a part of a join subquery and the result of that dead subquery was used to constitute (a part of) the top environment on ES. However, in a general case a dead subquery does not have to satisfy those conditions. Let's analyze an example. Suppose we have the view

view *NamesOfProfFaculty*
begin
  return *Professor*.(*works_in*.(*name* **as** *n*, *Faculty.fname* **as** *f*))
end;

which for each professor returns his/her name and the name of the faculty he/she works in (those subresults are named *n* and *f*, respectively). If we want to get professors' names via this view, we can ask the following query:

NamesOfProfFaculty.n

After macro-substitution it has the form:

(*Professor*.(*works_in*.(*name* **as** *n*, *Faculty.fname* **as** *f*))).*n* (ii)

The subquery *Faculty.fname* **as** *f* is dead just like the dead subqueries considered so far (it is the only dead part of query (ii)); we remove it:

(*Professor*.(*works_in*.(*name* **as** *n*))).*n* (iii)

However, a subquery that was not dead in (ii) turns out to be dead in (iii): the subquery *works_in* died, because the only subquery that had used its result (i.e., *Faculty*) has been removed as a component of the dead subquery.

Note that the dead subquery neither forms a join subquery or its result is used to build the top environment on ES, when the final dot operator (which is the last operator whose right subquery could use the result of *works_in*) is being evaluated. Such cases require further research so that our method could transform (iii) to

(*Professor*.(*name* **as** *n*)).*n*

A final remark: In some cases **where** can cause a subquery to be dead. An example is the case when the left subquery of **where** is a union of queries:

(*Q1* **union** *Q2*) **where** *Q3*

It may turn out that the final result of the whole query does not contain the result of *Q1* (or *Q2*) which makes the subquery dead. This also needs further research.

7 Conclusions

In the paper we have discussed in detail a special query optimization technique based on detecting and removing dead parts of queries through rewriting a query into a semantically equivalent form without dead parts. The method is based on static analysis of a given query in order to gather all the information that we need to detect dead subqueries. The only internal form of a query being optimized is a syntax tree.

In our research we have used the formal stack-based model, which allows us to describe precisely the semantics of query languages for a general object-oriented database model. In this setting we have described our approach to views and we have explained how we make use of the powerful technique of query modification (macro-substitution).

As we have pointed there are special cases of dead subqueries that are not covered by our method. They require further research.

References

1. Fernandez, M., Simeon, J., Wadler, P.: XML Query Languages: Experiences and Exemplars. www-db.research.bell-labs.com/user/simeon/xquery.html. (2000).
2. Cattel, R., Barry, D.: Object Data Standard 3.0. Morgan Kaufmann. (2000).
3. Ott, N., Horlander, K.: Removing Redundant Join Operations in Queries Involving Views. Information Systems 10(3). (1985) 279–288.
4. Płodzień, J.: Optimization Methods in Object Query Languages. Ph.D. Thesis. Institute of Computer Science, Polish Academy of Sciences. (2000).
5. Płodzień, J., Kraken, A.: Object Query Optimization through Detecting Independent Subqueries. Information Systems 25(8). (2000) 467–490.
6. Płodzień, J., Subieta, K.: Optimization of Object-Oriented Queries by Factoring Out Independent Subqueries. Institute of Computer Science, Polish Academy of Sciences. Report 889. (1999).
7. Płodzień, J., Subieta, K.: Static Analysis of Queries as a Tool for Static Optimization. Proc. of IDEAS. to appear (2001).
8. Stonebraker, M.: Implementation of Integrity Constraints and Views by Query Modification. Proc. of SIGMOD. (1975) 65–78.
9. Subieta, K.: Mapping Heterogeneous Ontologies through Object Views. Proc. of 3^{rd} Workshop on Engineering Federated Information Systems. (2000) 1–10.
10. Subieta, K., Beeri, C., Matthes, F., Schmidt, J.: A Stack-Based Approach to Query Languages. Proc. of 2^{nd} Intl. East-West Database Workshop. (1995) 159–180.
11. Subieta, K., Kambayashi, Y., Leszczyłowski, J.: Procedures in Object-Oriented Query Languages. Proc. of VLDB. (1995) 182–193.
12. Subieta, K., Płodzień, J.: Object Views and Query Modification. In: Barzdins, J., Caplinskas, A. (eds.): Databases and Information Systems. Kluwer Academic Publishers. (2001) 3–14.

The Impact of Buffering on Closest Pairs Queries Using R-Trees

Antonio Corral[1], Michael Vassilakopoulos[2], and Yannis Manolopoulos[3⋆]

[1] Department of Languages and Computation
University of Almeria, 04120 Almeria, Spain
acorral@ual.es

[2] Lab of Data Engineering, Department of Informatics
Aristotle University, 54006 Thessaloniki, Greece
mvass@computer.org

[3] Department of Computer Science
University of Cyprus, 1678 Nicosia, Cyprus
manolopo@ucy.ac.cy

Abstract. In this paper, the most appropriate buffer structure, page replacement policy and buffering scheme for closest pairs queries, where both spatial datasets are stored in R-trees, are investigated. Three buffer structures (i.e. single, hybrid and by levels) over two buffering schemes (i.e. local to each R-tree, and global to the query) using several page replacement algorithms (e.g. FIFO, LRU, 2Q, etc.) are studied. In order to answer K closest pair queries (K-CPQs, with $K \geq 1$) we employ recursive and non-recursive (iterative) branch-and-bound algorithms. The outcome of this study is the derivation of the outperforming configuration (in terms of buffer structure, page replacement algorithm and buffering scheme) for CPQs. In all cases, the savings in disk accesses is larger for a recursive algorithm than for a non-recursive one, in the presence of buffer space. Also, the global buffering scheme is more appropriate for small or medium buffer sizes for recursive algorithms, whereas the local scheme is the best choice for large buffers. If we use non-recursive algorithms, the global buffering scheme is the best choice in all cases. Moreover, LRU is the most appropriate page replacement algorithm for small or medium buffer sizes for both types of branch-and-bound algorithms. FIFO and LRU are the best choices for recursive algorithms and 2Q for the non-recursive ones, when the buffer is large enough.

1 Introduction

The use of buffers is very important in DBMSs, since it can improve the performance substantially (reading data from the disk is significantly more expensive than reading from a main memory buffer). There exist two basic research directions that aim at reducing the disk I/O activity and enhancing the system

⋆ On sabbatical leave from the Department of Informatics, Aristotle University, 54006 Thessaloniki, Greece. manolopo@csd.auth.gr throughput during query processing using buffers. The first one focuses on the availability of buffer pages at runtime by adapting memory management techniques for buffer managers used in operating systems to database systems [1,9, 13,15]. The second one focuses on query access patterns, where the query optimizer dictates the query execution plan to the buffer manager, so that the latter can allocate and manage its buffer accordingly [4,6,20].

The spatial selections, nearest neighbor searches and joins are considered the most important queries in spatial databases that are based on R-trees. R-trees [10] are multi-dimensional, height balanced tree structures for secondary storage, that handle objects by means of their Minimum Bounding Rectangles (MBRs). In [5] a new kind of spatial query, called K closest pairs query (K-CPQ), is presented. It combines join and nearest neighbor queries for discovering the K pairs ($K \geq 1$) of spatial objects from two datasets that have the K smallest distances between them (1-CPQ is treated as special case). Like a join query, all pairs of objects are candidates for the result. Like a nearest neighbor query, proximity metrics are the basis for pruning strategies and the final ordering.

The main objective of this work is to find the most appropriate buffer structure, page replacement policy and buffering scheme for CPQs, where both spatial datasets are indexed with R-trees. Based on experimental results, we draw conclusions about the importance of using an appropriate buffer management for the I/O performance of this kind of query. We present a comparative study, where several parameters (such as the buffer structure, page replacement algorithms, buffering schemes, buffer size in pages, number of pairs in the result K and the nature of indexed datasets) and corresponding values are considered.

The rest of this paper is organized as follows. In Sect- 2, we review the literature (CPQs using R-trees and buffering) and motivate the research topic under consideration. In Sect. 3, a brief description of the spatial access method (i.e. R-tree) and the branch-and-bound algorithms (i.e. recursive and non-recursive) for satisfying CPQs are presented. In Sect. 4, in order to study the effect of buffering in the performance of this kind of query, we examine combinations of buffer structures, page replacement algorithms and buffering schemes. Moreover, in Sect. 5, an extensive comparative performance study of CPQ algorithms over these alternative combinations is presented. Finally, in the last section, the conclusions on the contribution of this paper and future research plans are summarized.

2 Related Work and Motivation

In DBMSs, the buffer manager is responsible for operations in the buffer pool, including buffer space assignment to queries, replacement decisions and buffer reads and writes in the event of page faults. When buffer space is available, the manager decides about the number of pages that are allocated to an activated query. This decision may depend on the availability of pages at runtime (page replacement algorithms), or the access pattern of queries (nature of the query). A number of studies focus on adapting memory management techniques used in operating systems to database systems, such as FIFO, LRU, LFU, Gclock, etc. [1,9,13,15]. Other research efforts aim at determining the buffer requirements of queries based on their access patterns (the nature of the query) without considering the availability of buffer pages at runtime [4,6,20].

Since this paper is related to the research directions based on the nature of the query, we focus in the most representative papers about the buffer management on indices. In [18], an LRU buffer structure for indices was presented (OLRU), where the addressing space is logically partitioned into L independent regions, each managed by a local LRU chain. In [6] an extensible and dynamic priority-based hint mechanism was proposed to design an optimal replacement strategy by exploiting the predictable access pattern of indexing methods. An application on their hint mechanism was to design a hybrid replacement strategy, combining the LRU and MRU page replacement policies. There are several studies on spatial queries involving more than one R-tree, and most of them examine the use of buffering to reduce the I/O activity [3,5,7,11,12,17].

All the previous papers involved more than one R-tree for the query and used a buffer pool with LRU or FIFO replacement policy, but they did not justify the use of these policies. In other words, they did not examine several alternatives for the buffer structure, or for the page replacement strategies in order to reduce the disk activity. In this paper, our objective is to find the most appropriate buffer pool structure (i.e. single, hybrid and by levels) over two buffering schemes (i.e. local and global) and the best page replacement policy (e.g. FIFO, LRU, Gclock, etc.) for CPQs, where both spatial datasets are indexed by R-trees.

3 R-Trees and Algorithms for Closest Pairs Queries

3.1 R-Trees

R-trees [10] are hierarchical, height balanced data structures based on B$^+$-trees, used for the dynamic organization of k-dimensional geometric objects that are represented by k-dimensional MBRs. R-trees obey the following rules. Leaves reside on the same level and contain pairs of the form (R, O), where R is the MBR containing the object determined by the identifier O, spatially. Internal nodes contain pairs of the form (R, P), where P is a pointer to a child of the node and R is the MBR containing (spatially) the rectangles stored in this child. Also, internal nodes correspond to MBRs containing (spatially) the MBR of their children. An R-tree of class (m, M) has the characteristic that every node, except possibly for the root, contains between m and M pairs, where $m \leq \lceil M/2 \rceil$. If the root is not a leaf, it contains at least two pairs. Figure 1 depicts some rectangles on the right and the corresponding R-tree on the left. Dotted lines denote the bounding rectangles of the subtrees that are rooted in inner nodes.

Many R-tree variants have appeared in the literature. One of the most popular variations is the R*-tree [2], which follows a sophisticated node split technique and is considered to be the most efficient variant of the R-tree family. In this paper, we have chosen R*-trees to perform our experimental study, although in the sequel, the terms R-tree and R*-tree will be used interchangeably.

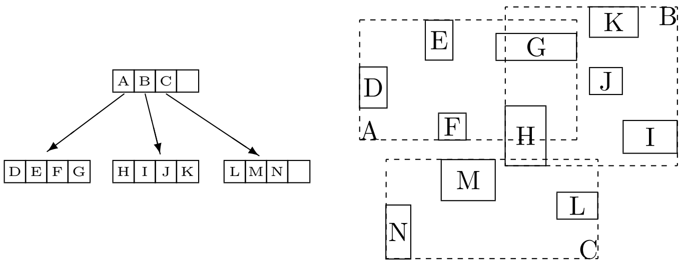


Fig. 1. An example of an R-tree

3.2 Algorithms for Closest Pairs Queries

A new spatial query was presented in [5], called K closest pairs query (K-CPQ). It combines join and nearest neighbor queries for discovering the K pairs ($K \geq 1$) of spatial objects from two datasets that have the K smallest distances between them. These queries are defines as follows.

1-CPQ. Assume two object datasets P and Q (where $P \neq \emptyset, Q \neq \emptyset$), stored in two R-trees, R_P and R_Q, respectively. Find the pair of objects p, $p \in P \times Q$, such that: $dist(p) \leq dist(p'), \forall p' \in (P \times Q - \{p\})$, where $dist$ is a Minkowski distance of the pairs of $P \times Q$.

K-CPQ. Assume two object datasets P and Q (where $P \neq \emptyset$, $Q \neq \emptyset$), stored in two R-trees, R_P and R_Q, respectively. Find the K ordered pairs of objects $p_1, p_2, \ldots, p_K, p_i \in P \times Q$, such that: $dist(p_1) \leq dist(p_2) \leq \ldots \leq dist(p_K) \leq dist(p'), \forall p' \in (P \times Q - \{p_1, p_2, \ldots, p_K\})$.

Metrics (MINIMINDIST, MINMAXDIST and MAXMAXDIST) and properties between two MBRs in the k-dimensional Euclidean space were proposed for the 1-CPQ and K-CPQ in [5] as bounds for the branch-and-bound (recursive and non-recursive) algorithms. The recursive branch-and-bound algorithm (with a synchronous traversal, following a depth-first search strategy) for processing the 1-CPQ between two sets of points stored in two R-trees with the same height can be described by the following steps:

CPQ1. Start from the roots of the two R-trees and set the minimum distance found so far, T, to ∞.

CPQ2. If you access a pair of internal nodes, then calculate the minimum of MINMAXDIST for all possible pairs of MBRs. If this minimum is smaller than T, then update T. Calculate MINMINDIST for each possible pair of MBRs. Propagate downwards recursively only for those pairs having MINMINDIST$\leq T$.

CPQ3. If you access two leaves, then calculate the distance of each possible pair of points. If this distance is smaller than T, then update T.

The non-recursive branch-and-bound algorithm (with a synchronous traversal, following a best-first search strategy using a minimum heap) for processing the 1-CPQ between two sets of points stored in two R-trees with the same height can be described by the following steps:

CPQ1. Start from the roots of the two R-trees, set T to ∞ and initialize the minimum heap.

CPQ2. If you access a pair of internal nodes, then calculate the minimum of MINMAXDIST for all possible pairs of MBRs. If this minimum is smaller than T, then update T. Calculate MINMINDIST for each possible pair of MBRs. Insert into the minimum heap those pairs having MINMINDIST$\leq T$.

CPQ3. If you access two leaves, then calculate the distance of each possible pair of points. If this distance is smaller that T, then update T.

CPQ4. If the minimum heap is empty, then stop.

CPQ5. Get the pair on top of the minimum heap. If this pair has MINMINDIST$>T$, then stop. Else, repeat the algorithm from CPQ2 for this pair.

The pseudo-code of the recursive and non-recursive algorithms can be found in the technical report [8]. Moreover, in order to process the K-CPQ, an extra structure that holds the K closest pairs is necessary. More details can be found in [5].

4 Buffer Management

DBMSs use indices to speed up query processing (e.g. various spatial databases use R-trees). Indices may partly reside in main memory buffers. This reduces response times. The buffering effect should be studied, since even a small number of buffer pages can substantially improve the global database performance. Our objective is to find the best structure of the buffer pool, the best page replacement algorithm and the best buffering scheme for the buffer manager in order to reduce the number of disk accesses for K-CPQs. We propose three structures of the buffer pool (i.e. single, hybrid and by levels) managed by a variety of page replacement algorithms (e.g. FIFO, LRU, etc.).

The buffer pool structure will be organized adopting two buffering schemes as depicted in Fig. 2. In the first scheme, the buffer pool is split in two parts, each one allocated locally to an R-tree (left part of Fig. 2). We call it, thus, a Local buffering scheme. In the second one, the buffer pool is allocated globally to the query (right part of Fig. 2), giving rise to a Global buffering scheme.

In [9] a systematic description of replacement algorithms was presented for a single buffer structure. The FIFO (First-In First-Out) algorithm replaces the oldest page, even if its reference frequency gives the priority to the youngest page. The LFU schema (Least Frequently Used) replaces the page with the lowest reference frequency. Gclock consists of a circular decrementing of the reference counters until 0 is reached. When a buffer fault occurs, the first page having a counter equal to 0 is replaced. The LRU (Least Recently Used) algorithm gives the priority to the most recently used page, replacing the page that was the least recently used. MRU (Most Recently Used) is the opposite of LRU and replaces the page that was the most recently used. The LRU/2 is a particular case of LRU/K, proposed in [15] for $K = 2$, replacing the page whose penultimate (second-to-last) access is the least recent among all penultimate accesses. LRD

(Least Reference Density) is not a page replacement algorithm based on page ages, but on its reference density (reference probability) from the first time that a page was accessed. The page replacement algorithm LRD rejects from the buffer the page with the minimum reference density. Finally, in [16] a page replacement algorithm for spatial databases, called LRD-Manhattan, was proposed as a variation of LRD.

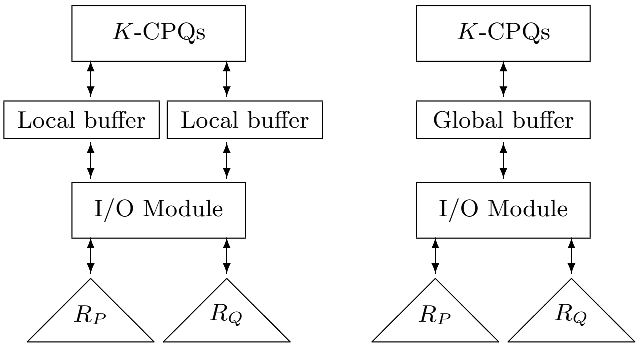


Fig. 2. Local and Global buffering schemes

The most representative methods for the hybrid buffer structure are the techniques called 2Q and FIFO_LRU. The 2Q algorithm divides the buffer pool in two areas: the hot area managed as an LRU queue and the cold area maintained as a FIFO queue [13]. On the first reference of a page, 2Q places it in the cold area (FIFO). If the page is re-referenced while in the cold area, then it is moved to the hot area (LRU). Evidently, if a page is not re-referenced while in the cold area, it is rejected from the buffer. In order to solve the "correlated references" problem, 2Q divides the cold area in two parts, one for pages and another for page identifiers. The FIFO_LRU technique works in the same way as 2Q, but the hot area is implemented as a FIFO replacement algorithm and the cold area is managed with an LRU policy [1].

Here, we present a buffer structure linked to each R-tree based on its height, h, for solving K-CPQs. This means that the buffer pool is split in h independent areas. For each R-tree level we allocate a number of pages according to its minimum fan-out factor m and its height, with the exception of the root, for which we allocate only one page. We create this buffer structure in a bottom-up way, trying to set a distribution of pages per level as fair as possible (root level=level $h-1 : m^0$, level $h-2 : m^1$, level $h-3 : m^2$, ..., level $1 : m^{h-2}$, leaf level=level $0 : m^{h-1}$). In the case of K-CPQs, pages at lower levels are very important for the branch-and-bound algorithms. Besides, we manage these h independent areas using a specific page replacement algorithm, for example LRU (LRU_L=LRU by levels), or FIFO (FIFO_L=FIFO by levels).

5 Experimentation

This section summarizes the results of an extensive experimentation that aims at measuring and evaluating the behavior of the recursive and non-recursive branch-and-bound algorithms for K-CPQs using different structures, schemes, policies and buffer sizes. We ignore the effect of path-buffer [5], since it offers more advantages to the recursive algorithms, regardless of the page replacement policy.

For our experiments, we have built several R*-trees [2] using the following datasets: (a) a real dataset from the Sequoia project [19] consisting of 62.536 points that represent specific country sites of California (Real), (b) a point dataset produced from the real one by moving randomly every point (Real$'$) and (c) two datasets of cardinality 62.536 points, which completely overlap and follow uniform and skewed distributions [5]. All experiments have run on a Linux workstation with 128 Mb of main memory and several Gb of secondary storage, making use of GNU C++ compiler. The page size was 1 Kb, resulting to a maximum R*-tree node capacity $M = 21$ (minimum capacity was set to $m = M/3 = 7$, a reasonable choice according to [2]). The quantity counted in all experiments was the number of disk accesses required to perform the K-CPQs.

5.1 K-CPQ Algorithms Using a Local Buffering Scheme

We now proceed to the performance comparison of the recursive and non-recursive branch-and-bound algorithms for K-CPQs using a Local buffering scheme in order to investigate the best page replacement policy and buffer structure. We used a buffer pool, B, with varying size from 0 to 512 pages, dedicating different portions of B to each R*-tree. The datasets joined were Real/Real$'$ and Uniform/Skewed. However, in the sequel we focus on Real/Real$'$ data sets, since both cases gave very similar trends.

First of all, for the hybrid structure in the Local or Global buffering scheme, we have performed several experiments with different B values ($B/2$ for each R*-tree) using recursive and non-recursive algorithms to derive the best page distribution for the hot and cold regions in the buffer. If B_P is the number of pages in the local buffer of the R*-tree R_P, the best configuration was <Hot, Cold> = $< B_P/2,\ B_P/2 >$. Moreover, for the Local buffering scheme, we have assigned a varying number of pages to each R*-tree, and the best distribution of the buffer was to assign more pages to the largest R*-tree, whatever the type (recursive, or non-recursive) of algorithm used. Since in our experimentation we have point datasets with identical cardinalities, $(B/2, B/2)$ was the best configuration [8].

We have run experiments using different page replacement policies over the three buffer pool structures. The best policies for the recursive algorithms were FIFO and LRU in case of small buffers (e.g. $B \leq 64$), but in case of large buffers (e.g. $B \geq 128$) LRU_L was slightly better than FIFO and LRU. FIFO and LRU were better than LFU, Gclock, MRU, LRU/2 and LRD, because recursion favors the youngest and most recently used pages in the backtracking phase and this behavior is slightly improved in case of large buffers organized by levels (FIFO_L and LRU_L). On the other hand, for the non-recursive algorithms and small buffers (e.g. $B \leq 64$), FIFO and LRU were again the best policies, whereas for large buffers (e.g. $B \geq 128$) 2Q was slightly better than FIFO and LRU. In this case, we did not use recursion and the organization of the buffer pool in two regions (i.e. hot and cold) provided a good performance, when the search strategy was best-first implemented through a heap of minimums and the buffer was large enough. For instance, for the recursive 1-CPQ, using a single buffer structure, MRU was 35% worse with respect to the LRU. Under these conditions, Gclock was 4% worse with respect to LRU, LFU 35% worse than FIFO, LRU/2 20% worse than LRU, and LRD 32% worse than FIFO. These behaviors are depicted in Fig. 3, where different page replacement policies are compared, using the recursive algorithm for 1-CPQ in a single buffer structure. Besides, if we include a large buffer (e.g. $B = 512$) with the single structure and the LRU policy, the savings in I/O operations were 73% for the recursive algorithm and 68% for the non-recursive one with respect to the absence of buffer space ($B = 0$). For the non-recursive algorithm the results were very similar.

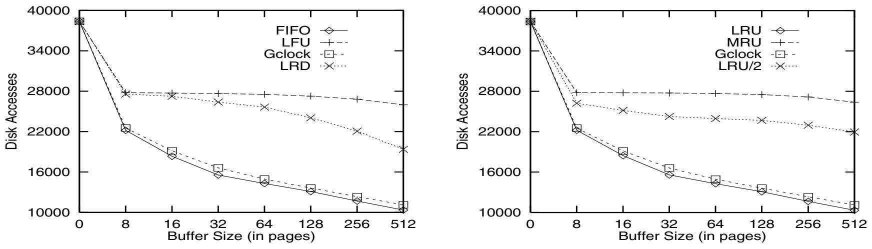


Fig. 3. The performance of the 1-CPQ recursive algorithm for various page replacement policies and a single buffer structure, as a function of the buffer size

For the recursive and non-recursive algorithms, in Fig. 4 we illustrate the performance of the 1-CPQ recursive (left) and non-recursive (right) algorithms for various page replacement policies, as a function of the buffer size. It can be seen that the two charts follow the same trend. When the buffer size is small (e.g. $B \leq 64$), the single structure with LRU policy is the best (with 6% and 5% savings for LRU in comparison with 2Q, for recursive and non-recursive algorithms, respectively), the second is the hybrid and the third one is by levels. However, in case of large buffers (e.g. $B \geq 128$) the difference is almost negligible for all page replacement policies, although LRU_L and 2Q are slightly better that the other for the recursive and non-recursive algorithms, respectively.

The results of the recursive K-CPQ algorithm for a given buffer size (e.g. $B = 512$) showed that the best behavior was for LRU_L with a 0.5% improvement over LRU (for all K values), whereas the worst results appeared in the case of the hybrid structure (2Q and FIFO_LRU). For the non-recursive algorithm with the same number of buffer pages ($B = 512$), the best behavior was for 2Q with a

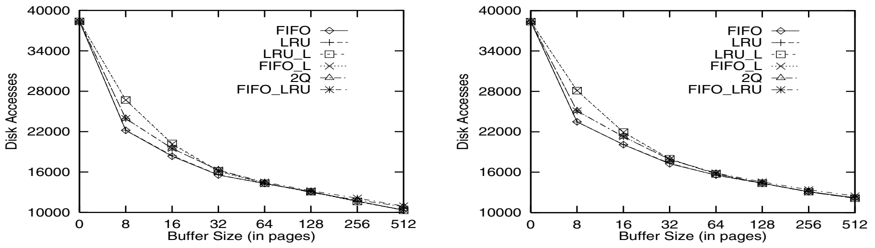


Fig. 4. The performance of the of 1-CPQ recursive (left) and non-recursive (right) algorithms for various page replacement policies, as a function of the buffer size

0.6% improvement over LRU (for all K values), whereas the worst results were for FIFO_LRU [8].

In the case of 1-CPQ, the recursive algorithm presents 10% excess of I/O activity in comparison to the non-recursive one with the same page replacement policy (LRU), as can be noticed by the gap between the two lines in the left part in the Fig. 5. The gap for K-CPQ is bigger when the K value is incremented; it is 25% bigger for $K \leq 10000$, but it reaches 45% when $K = 100000$ (see the right part of Fig. 5). Besides, by increasing K values (1..100000), the performance of the recursive algorithm is not significantly affected; with a buffer of 512 pages and the best page replacement algorithm there is an extra cost of 2%. On the other hand, this extra cost is about 39% for the non-recursive algorithm using the same buffer characteristics. If we do not have any buffer space ($B = 0$), then increasing K implies an additional cost of 33% for the recursive algorithm and 16% for the non-recursive one. Moreover, the recursive variant demonstrates savings in the range 73%-82%, when K increases (1..100000) and a buffer of 512 pages is used, in comparison to the no buffer case ($B = 0$). The non-recursive algorithm under the same buffer setup results in savings from 68% to 57%.

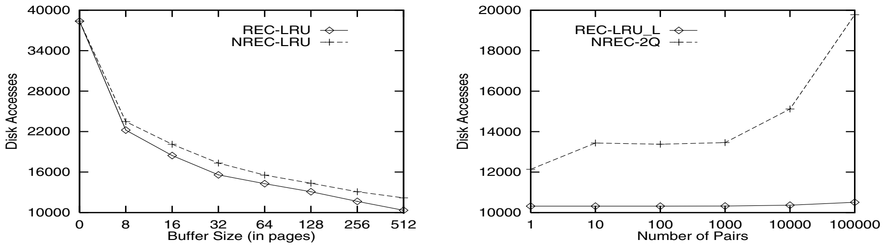


Fig. 5. The performance of the 1-CPQ (left) and the K-CPQ (right) recursive (REC) and non-recursive (NREC) algorithms using the best page replacement policies and $B = 512$, as a function of the buffer size

In Fig. 6, the percentage of I/O cost savings (induced by the use of buffer size $B > 0$ in contrast to not using any buffer) of the K-CPQ recursive algorithm with LRU_L policy (left) and non-recursive algorithm with 2Q policy (right) is depicted. For the recursive algorithm, the percentage of savings grows as buffer sizes increase, for all K values, although it is bigger for $K = 100000$. The behavior of non-recursive algorithm is slightly different. When the buffer becomes larger, the percentage of savings also increases, but when we fix the buffer size, the increase of K causes a decrease in the percentage of savings. From all these results, we notice that the influence of buffering for a Local scheme is more important for the recursive algorithm than for the non-recursive one.

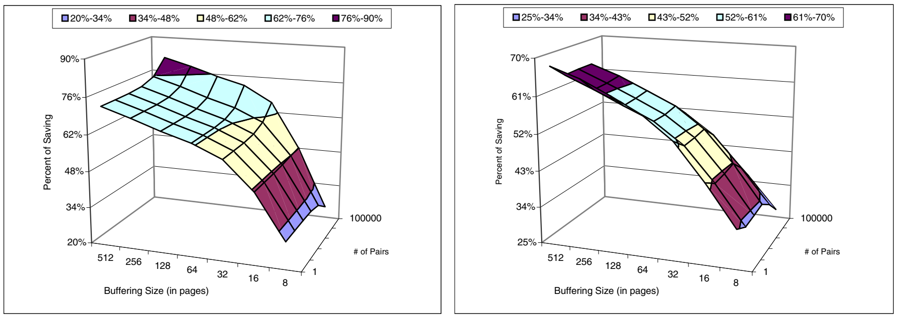


Fig. 6. The I/O cost savings of the K-CPQ recursive algorithm with LRU_L policy (left) and non-recursive algorithm with 2Q policy (right), as a function of B and the cardinalities of the data sets

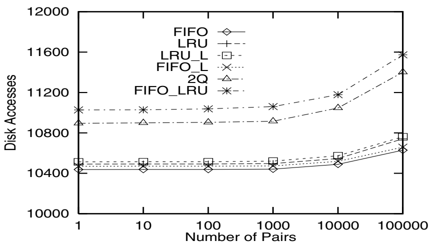
5.2 *K*-CPQ Algorithms Using a Global Buffering Scheme

For the Global buffering scheme, we have used the same parameters as for the Local one in order to investigate the best page replacement policy and buffer structure. In particular we used: (a) several replacement algorithms (FIFO, LRU, LRU_L, FIFO_L, 2Q and FIFO_LRU) for the three buffer structures, (b) the same number of pages for the buffer (B varying from 0 to 512 pages), and (c) the recursive and non-recursive algorithms for K-CPQ with K varying from 1 to 100000.

We have performed experiments with 1-CPQ using several replacement algorithms in the Global buffering scheme. When the buffer size was small or medium (e.g. $B \leq 128$), the single structure with LRU policy was the best (with 3% savings with respect to 2Q, for recursive and non-recursive algorithms), the second was the hybrid and the third one was by levels. Again, when the buffer was large (e.g. $B \geq 256$) the difference was almost negligible for all page replacement policies, although FIFO and 2Q were slightly better than the other ones for the recursive and non-recursive algorithms, respectively [8].

In the left part of Fig. 7, we depict the performance of the recursive K-CPQ algorithm for a given buffer size (e.g. $B = 512$). The best behavior is for FIFO with savings of 0.6% in relation to the LRU (for all K values), and the worst results are again for the hybrid structure (2Q and FIFO_LRU). On the other hand, the results of the non-recursive K-CPQ algorithm are illustrated in the right part of Fig. 7 for the same buffer size ($B = 512$). The best behavior arises for 2Q with savings of 0.6% in relation to LRU (for all K values), and the worst results are for FIFO_LRU.

For 1-CPQ, the buffering increased the performance of the recursive algorithm by 9% in comparison to the non-recursive one with the same page replacement policy (LRU). For K-CPQ, when the K value was incremented, this improvement was 26% approximately for $K \leq 10000$ and 47% for $K = 100000$. Besides, for increasing K values, the I/O cost of the recursive algorithm was not significantly affected, when we had a buffer of 512 pages and the best page replacement algorithm had only an extra cost of 2%. On the other hand, this extra cost was about 39% for the non-recursive algorithm using the same buffer characteristics. Moreover, the recursive variants demonstrated savings in the range 73%-81% as K increased, between the case of a 512 pages buffer and the case no buffer at all ($B = 0$). The non-recursive algorithm, under the same buffer setup, resulted in 68%-57% savings. In general, these results were very similar to the Local buffering scheme ones [8].



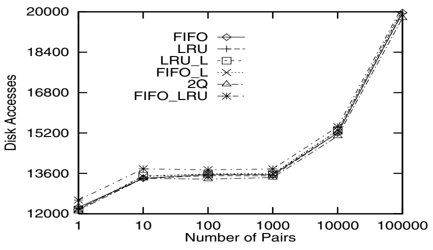
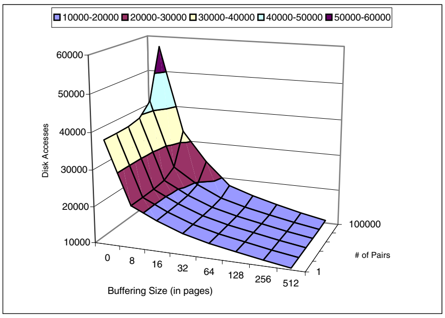


Fig. 7. The performance of the K-CPQ algorithm for different page replacement policies as a function of the buffer size for recursive (left) and non-recursive (right) algorithms and $B = 512$ pages

In Fig. 8, we see the performance of K-CPQ recursive and non-recursive algorithms as a function of buffer size ($B \geq 0$) with LRU policy. For the recursive algorithm, when $B \geq 32$, the savings in terms of disk accesses are large and almost the same for all K values. However, the savings are considerably less when $B \leq 16$, whereas for $K = 100000$ and $B = 0$ we can notice a characteristic peak. For the non-recursive algorithm, the savings trend is similar to the recursive one, but for high K values these savings become considerably less than the recursive one. For instance, if we have available enough buffer space, the recursive algorithm is the best alternative, because it provides an average I/O savings of 20% in respect to the non-recursive one for K-CPQ using LRU. For all these results, we notice that the influence of buffering for a Global scheme is more important for the recursive algorithm than for the non-recursive one in the K-CPQs, when we have enough buffer space. It is the same conclusion to that for the Local buffering scheme.



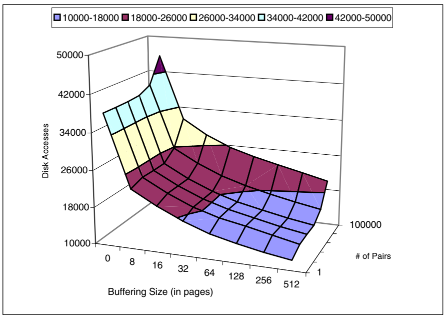


Fig. 8. The performance of K-CPQ recursive (left) and non recursive (right) algorithms with LRU policy, as a function of the buffer size and the cardinality of the data sets

5.3 Comparison of the Buffering Schemes for K-CPQ

Table 1 contains the results of an exhaustive comparison of the Local and Global buffering schemes, using the best buffer structure and page replacement algorithms for each of them. These results concern the performance of K-CPQs ($K \geq 1$) using the recursive (REC) and non-recursive (NREC) algorithms.

Table 1. Comparison of the Local and Global buffering schemes

Buffer Size	8	16	32	64	128	256	512
REC	**G** (LRU)	**G** (LRU)	**L** (LRU)	**G** (LRU)	**G** (LRU)	**G** (FIFO)	**L** (LRU_L)
NREC	**G** (LRU)	**G** (LRU)	**G** (LRU)	**G** (LRU)	**G** (2Q)	**G** (2Q)	**G** (2Q)

From this table (where L and G stand for Local and Global, respectively), we deduce that the Global buffering scheme is the best alternative in most cases, except for $B = 32$ and $B = 512$ for the recursive algorithm where the Local scheme prevails. The difference between the Global and Local schemes is around 1%-2% in terms of disk accesses for all cases. Since the difference is small, we suggest to use the Global buffering scheme, because, in this case, the buffer manager may: (a) include and handle more than two R-trees in the same buffer area, (b) give priority to a specific R-tree, (c) manage and assign dynamically more pages to one R-tree and (d) introduce global optimization techniques.

Besides, LRU is the most appropriate page replacement algorithm with a single buffer structure when the buffer size is small or medium. On the other hand, when the buffer is large the best alternatives are FIFO (single structure) and LRU_L (structure by level) for the recursive algorithm and 2Q (hybrid structure) for the non-recursive one. Since the difference between LRU and the other winner page replacement algorithms (FIFO, LRU_L and 2Q) is in the range 1%-2%, we suggest to use LRU as the policy with the best overall stable performance.

6 Conclusions and Future Work

Efficient processing of closest pairs queries (K-CPQs with $K \geq 1$) is of great importance in a wide area of applications like spatial databases, GIS, image databases, etc. Buffering is very important in DBMSs, because it improves the performance considerably (since reading from disk is orders of magnitude more expensive than reading from a buffer). In this paper we have examined the most important factors that affect the performance in the presence of a buffer. These are: the buffer structure, the page replacement algorithm, and the buffering scheme. From the experimentation we deduce the following conclusions:

– The I/O savings for the recursive algorithm are larger than that of the non-recursive one for K-CPQ when we have enough buffer space. The reason is that the use of recursion in a depth-first way is affected by the buffering scheme more than the case of a best-first search strategy implemented through a heap of minimums.
– With a fixed buffer size, increasing the number K of pairs in a CPQ for the recursive algorithm results in a negligible extra cost with respect to the additional cost for the non-recursive one.
– The Global buffering scheme is more appropriate when the buffer size is small or medium for the recursive algorithm, while the Local scheme is the best choice for large buffers. On the other hand, if we use the non-recursive algorithm, the Global buffering scheme is the best alternative for all cases.
– LRU is the most appropriate page replacement algorithm with a single buffer structure when the buffer size is small or medium, whatever the type (recursive, or non-recursive) of algorithm for K-CPQs. On the other hand, when the buffer is large, then the best alternatives are FIFO (single structure) and LRU_L (structure by levels) for the recursive algorithm and 2Q (hybrid structure) for the non-recursive one.

Future research may include:

– Study of alternative choices for the buffer structure, page replacement algorithm and buffering scheme in the Self-CPQ and Semi-CPQ [5], which are extensions of 1-CPQ and K-CPQ.
– Consideration of other spatial data structures and multi-dimensional data.
– Development of a cost model, taking into account the effect of buffering to analyze the number of disk accesses required for K-CPQs for R*-trees (along the same lines as in [14], where a cost model for range queries in R-trees has been developed).

References

1. W. Bridge, A. Joshi, M. Keihl, T. Lahiri, J. Loaiza and N. MacNaughton: "The Oracle Universal Server Buffer", *Proc. 23rd VLDB Conf.*, pp.590-594, Athens, Greece, 1997.

2. N. Beckmann, H.P. Kriegel, R. Schneider and B. Seeger: "The R*-tree: and Efficient and Robust Access Method for Points and Rectangles", *Proc. 1990 ACM SIGMOD Conf.*, pp.322-331, Atlantic City, NJ, 1990.
3. T. Brinkhoff, H.P. Kriegel and B. Seeger: "Efficient Processing of Spatial Joins Using R-Trees". *Proc. 1993 ACM SIGMOD Conf.*, pp.237-246, Washington, DC, 1993.
4. H.T. Chou and D.J. DeWitt: "An Evaluation of Buffer Management Strategies for Relational Database Systems", *Proc. 11th VLDB Conf.*, pp.127-141, Stockholm, Sweden, 1985.
5. A. Corral, Y. Manolopoulos, Y. Theodoridis and M. Vassilakopoulos: "Closest Pair Queries in Spatial Databases", *Proc. 2000 ACM SIGMOD Conf.*, pp.189-200, Dallas, TX, 2000.
6. C.Y. Chan, B.C. Ooi and H. Lu: "Extensible Buffer Management of Indexes", *Proc. 18th VLDB Conf.*, pp.444-454, Vancouver, Canada, 1992.
7. A. Corral, M. Vassilakopoulos and Y. Manolopoulos: "Algorithms for Joining R-Trees and Linear Region Quadtrees", *Proc. 6th SSD Conf.*, pp.251-269, Hong Kong, China, 1999.
8. A. Corral, M. Vassilakopoulos and Y. Manolopoulos: "The Impact of Buffering on Closest Pairs Queries using R-trees", *Technical Report*, Dept. of Informatics, Aristotle University of Thessaloniki, February 2001.
9. W. Effelsberg and T. Harder: "Principles of Database Buffer Management", *ACM Transactions on Database Systems, Vol.9, No.4*, pp.560-595, 1984.
10. A. Guttman: "R-trees: A Dynamic Index Structure for Spatial Searching", *Proc. 1984 ACM SIGMOD Conf.*, pp.47-57, Boston, MA, 1984.
11. Y.W. Huang, N. Jing and E.A. Rundensteiner: "Spatial Joins Using R-trees: Breadth-First Traversal with Global Optimizations", *Proc. 23rd VLDB Conf.*, pp.396-405, Athens, Greece, 1997.
12. G.R. Hjaltason and H. Samet: "Incremental Distance Join Algorithms for Spatial Databases", *Proc. 1998 ACM SIGMOD Conf.*, pp.237-248, Seattle, WA, 1998.
13. T. Johnson and D. Shasha: "2Q: a Low Overhead High Performance Buffer Management Replacement Algorithm", *Proc. 20th VLDB Conf.*, pp.439-450, Santiago, Chile, 1994.
14. S.T. Leutenegger and M.A. Lopez: "The Effect of Buffering on the Performance of R-Trees". *Proc. ICDE Conf.*, pp.164-171, Orlando, FL, 1998.
15. E.J. O'Neil, P.E. O'Neil and G. Weikum: "The LRU-K Page Replacement Algorithm for Database Disk Buffering", *Proc. 1993 ACM SIGMOD Conf.*, pp.297-306, Washington, DC, 1993.
16. A. Papadopoulos and Y. Manolopoulos: "Global Page Replacement in Spatial Databases", *Proc. DEXA'96, Conf.*, pp.855-864, Zurich, Switzerland, 1996.
17. A. Papadopoulos, P. Rigaux and M. Scholl: "A Performance Evaluation of Spatial Join Processing Strategies", *Proc. 6th SSD Conf.*, pp.286-307, Hong Kong, China, 1999.
18. G.M. Sacco: "Index Access with a Finite Buffer", *Proc. 13th VLDB Conf.*, pp.301-309, Brighton, England, 1987.
19. M. Stonebraker, J. Frew, K. Gardels and J. Meredith: "The Sequoia 2000 Benchmark", *Proc. 1993 ACM SIGMOD Conf.*, pp.2-11, Washington, DC, 1993.
20. G.M. Sacco and M. Schkolnick: "A Mechanism for Managing the Buffer Pool in a Relational Database System Using the Hot Set Model", *Proc. 8th VLDB Conf.*, pp.257-262, 1982.

Enhancing an Extensible Query Optimizer with Support for Multiple Equivalence Types

Giedrius Slivinskas and Christian S. Jensen

Department of Computer Science, Aalborg University, Denmark
`http://www.cs.auc.dk/~{giedrius|csj}`

Abstract. Database management systems are continuously being extended with support for new types of data and advanced querying capabilities. In large part because of this, query optimization has remained a very active area of research throughout the past two decades. At the same time, current commercial optimizers are hard to modify, to incorporate desired changes in, e.g., query algebras or transformation rules. This has led to a number of research contributions aiming to create extensible query optimizers, such as Starburst, Volcano, and OPT++.
This paper reports on a study that has enhanced Volcano to support a relational algebra with added temporal operators, such as temporal join and aggregation. These enhancements include the introduction of algorithms and cost formulas for the new operators, six types of query equivalences, and accompanying query transformation rules. The paper describes extensions to Volcano's structure and algorithms and summarizes implementation experiences.

1 Introduction

Query optimization has remained subject to active research for more than twenty years. Much research has aimed at enhancing existing optimization technology to enable it to support the requirements, such as for new types of data and queries, of the many and new types of application areas, to which database technology has been introduced over the years. However, current commercial optimizers remain hard to extend and modify when new operators, algorithms, or transformations have to be added, or when cost estimation techniques or search strategies have to be changed [4]. As a result, the last decade has witnessed substantial efforts aiming to develop extensible query optimizers that would make such changes easier. Representative examples of extensible query optimizers include Starburst [8], Volcano [7], and OPT++ [11].

This paper reports on a specific study that has enhanced the Volcano extensible query optimizer to support a relational algebra with temporal operators such as temporal join and aggregation [15]. In addition to new operators, cost formulas, selectivity-estimation formulas, and transformation rules, the algebra offers systematic support for order preservation and duplicate removal and retention for all queries, as well as for coalescing for temporal queries (in coalescing, several tuples with adjacent time periods and otherwise identical attribute values are merged into one). To support order, relations are defined as lists, and six kinds of relation equivalences are defined – two relations can be equivalent as lists, multisets, and sets, and two temporal relations can be snapshot-equivalent as lists, multisets, and sets. We report on the design decisions and implementation experiences, and we evaluate Volcano's extensibility.

An important goal of the algebra is to offer a foundation for a layered temporal DBMS that may evaluate temporal queries faster than do current DBMSs. The latter do not have efficient algorithms for expensive temporal operations such as temporal aggregation, while such operations can be evaluated efficiently at the user-application level by algorithms that use cursors to access the underlying data [16].

New algorithms can be added to a DBMS via, e.g., user-defined routines in Informix [2,9] or PL/SQL procedures in Oracle [13], but these methods currently do not allow to define functions that take tables as arguments and return tables [10]; nor do they allow to specify transformation rules, cost formulas, and selectivity-estimation formulas for the new functions. Because of these limitations, a middleware component with query processing capabilities was introduced, which divides the query processing between itself and the underlying DBMS [16]. Intermediate relations can be moved between the middleware and the DBMS by the help of transfer operators.

To adequately divide the processing, the middleware has to take optimization decisions – for this purpose, we employ the Volcano extensible optimizer. Use of a separate middleware optimizer allows us to take advantage of transformation rules and cost and selectivity-estimation formulas specific to the temporal operators.

This paper summarizes design issues and experiences from the implementation of the optimizer. While the addition of new temporal operators, their cost and selectivity-estimation formulas, and transformation rules could be done using the extensibility framework provided by Volcano, adding support for multiple types of equivalences between relations required changes in Volcano structures, and in its search-space generation and plan-search algorithms.

To our knowledge, no existing extensible query optimizers systematically support sets, multisets, and lists. Sorting is treated differently than the common operators, such as selection or join, and it usually is considered in the query optimization only after the search space of possible query plans has been generated. However, particularly due to recent introduction and increasing use of `TOP N` and `BOTTOM N` predicates in queries [3], sorting could be exploited better in query optimization if considered during the search-space generation.

The paper is structured as follows. In Sect. 2, we present Volcano's architecture. Section 3 describes the enhancements to Volcano that were necessary to support the algebra introduced above. The algebraic framework is described first, with a focus on the parts that posed challenges to Volcano, and the modifications are described next. Section 4 summarizes the implementation experiences and evaluates the extensibility of Volcano. Section 5 covers related work, and Sect. 6 concludes the paper.

2 Description of Volcano

The Volcano Optimizer Generator is a software program for generating extensible query optimizers. The input to the program is a query algebra: operators, their implementation algorithms (physical operators), transformation rules, and implementation rules. Transformation rules specify equivalent logical expressions, and implementation rules specify which algorithms implement which operators. The output is an optimizer, which takes a query in the given algebra as input and returns a physical expression (an expression of algorithms) representing the chosen query evaluation plan. The optimizer implementor's tasks include the specification of the input and the coding of the support functions – such as the selectivity estimation – for operators and rules.

2.1 Two Stages of Query Optimization

The Volcano optimizer optimizes queries in two stages. First, the optimizer generates the entire search space consisting of logical expressions generated using the initial query plan (to which the query is mapped to) and the set of transformation rules. The search space is represented by a number of equivalence classes. An equivalence class may contain one or more logically equivalent expressions, also called elements; each of these includes an operator, its parameter (for example, predicates for the selection), and pointers to its inputs (which are also equivalence classes).

Consider a simple example query, which performs a join on the `EmpID` attribute of `POSITION` and `SALARY` relations. Its one possible initial plan is shown in Fig. 1(a) and its search space is shown in Fig. 1(b). The elements of classes 1 and 2 represent logical expressions returning partial results of the query, i.e., the operators retrieving, respectively, the `POSITION` and `SALARY` relations. The elements of class 3 represent logical expressions returning the result of the complete query; either the first or the second element may be used. Essentially, the given search space represents only two plans which differ in the order of the join arguments.

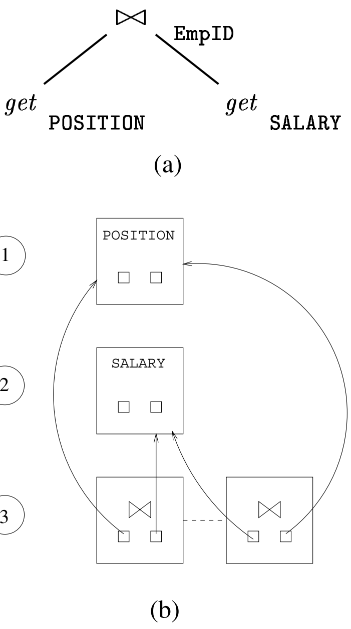


Fig. 1. A Query Plan and its Search Space

During the second stage of Volcano's optimization process, the search for the best plan is performed. Here, the implementation rules are used to replace operators by algorithms, and the costs of diverse subplans are estimated. For the given query, the number of plans to be considered is greater than two, because the relations may be retrieved by using either full scan or index scan, and the join may be implemented by, e.g., nested-loop, sort-merge, or index join. One possible evaluation plan is to scan both relations and perform a nested-loop join.

The following two sections briefly describe the search-space generation and the plan-search algorithms; for more detail, we refer to [7].

2.2 Stage One: Search-Space Generation

The search-space generation is performed by the *Generate* function. Initially, one element is created for each operator in the original query expression, and then *Generate* is invoked on the top element.

The *Generate* function repeatedly applies transformation rules to the given element, choosing among the applicable rules that have not so far been applied to the element. The application of a transformation rule may trigger the creation of new elements and classes; for each newly generated element, the *Generate* function is invoked.

For the query in Fig. 1(a), the search space is generated as follows. Initially, three elements representing the three query-tree operators are created (the first elements of equivalence classes 1–3 in Fig. 1(b)). Then, the *Generate* function is invoked for the first element of class 3, which, in turn, invokes *Generate* for the first elements of classes 1 and 2. The latter two *Generate* calls do not do anything because no rules apply to the elements of class 1 and 2. For the first element of class 3, however, the join commutativity rule is applied, and a second element pointing to switched join arguments is added to class 3. Then, the *Generate* function is invoked on the new element of class 3, but no new elements are generated: the join commutativity rule is applied again, but its resulting right-hand element already exists in the search space.

2.3 Stage Two: Plan Search

When searching for a plan, the Volcano optimizer employs dynamic programming in a top-down manner, and it uses the *FindBestPlan* function recursively.

First, the optimizer invokes the *FindBestPlan* function for the first element of the top equivalence class – e.g., class 3 in Fig. 1(b) – and a cost limit of infinity (the cost limit can be lower in subsequent calls to the function). If all elements of the class containing the argument element have already been optimized, no further optimization for the element is necessary: if the plan has been found and its cost is lower than the cost limit, it is returned, if not – NULL is returned. Otherwise, optimization has to be performed.

During the optimization, for each algorithm implementing the top operator (in our case, join), *FindBestPlan* is recursively invoked on the inputs to the algorithm. If optimization of the inputs is successful, the plan with the algorithm yielding the cheapest expected cost is chosen as the best plan. Then, *FindBestPlan* is recursively invoked for each equivalent logical expression (in our case, for the second element in equivalence class 3) to see if a better plan can be found. In case a better plan is found, it is saved in memory as the best one.

3 Enhancement of Volcano

The implementation of the algebra and its accompanying transformation rules introduces several concepts that did not exist previously in Volcano; these new concepts are described in Sect. 3.1. Sections 3.2 and 3.3 concern the actual implementation.

3.1 Algebra and Multi-equivalence Transformation Rules

First, we overview the architecture for which the algebra has been designed. Next, we describe the actual algebra, the accompanying transformation rules, and their applicability, focusing on the new concepts. Finally, we outline the challenges that these new concepts pose to Volcano.

Architecture. The temporally extended relational algebra [15] has been designed for an architecture consisting of a middleware component and an underlying DBMS. Expensive temporal operations such as temporal aggregation do not have efficient algorithms in the DBMS, but can be evaluated efficiently by the middleware, which uses a cursor to access DBMS relations [16]. Consequently, query processing is divided between the middleware and the DBMS; the main processing medium is still the DBMS, but the middleware is used when this can yield better performance.

Algebra. The algebra differs from the conventional relational algebra in several aspects. First, it includes temporal operators such as temporal join and temporal aggregation. Next, it contains two transfer operators that allow to partition the query processing between the middleware and the DBMS. Finally, the algebra provides a consistent handling of duplicates and order at logical level, by treating duplicate elimination and sorting as other logical operators and by introducing six types of relation equivalences.

Two relations are equivalent (1) as lists if they are identical lists ($\equiv_L$); (2) as multisets if they are identical multisets taking into account duplicates, but not order ($\equiv_M$); and (3) as sets if they are identical sets, ignoring duplicates and order ($\equiv_S$). Two temporal relations are snapshot-list ($\equiv_L^S$), snapshot-multiset($\equiv_M^S$), or snaphot-set equivalent ($\equiv_S^S$), if their snapshots (projections at a given point in time) are equivalent as lists, multisets, or sets.

Figure 2 shows two temporal relations (relations having two attributes indicating a time period), POSITION and SALARY. We assume a closed-open representation for time periods and assume the time values for T1 and T2 denote months during some year. For example, Tom was occupying position Pos1 from February to August (not including the latter).

POSITION

PosID	EmpID	Name	T1	T2
Pos1	1	Tom	2	8
Pos2	2	Jane	3	8

SALARY

EmpID	Amount	T1	T2
1	100K	2	6
1	120K	6	9
2	110K	3	8

Result

EmpID	Name	PosID	Amount	T1	T2
1	Tom	Pos1	100K	2	6
1	Tom	Pos1	120K	6	8
2	Jane	Pos2	110K	3	8

Fig. 2. Relations POSITION and SALARY, and the Result of Temporal Join

A temporal join is a regular join, but with a selection on the time attributes, ensuring that the joined tuples have overlapping time periods; Figure 2 shows the result of temporal join on the EmpID attribute of the POSITION and SALARY relations.

Transformation Rules. Six types of equivalences lead to six types of transformation rules, since a transformation rule may satisfy several of the six equivalences. Let us consider two rules for temporal join, $\bowtie^T$. For a given rule, we always specify the strongest equivalence type that holds; the ordering of equivalence types is given in Fig. 3. The join commutativity rule $r_1 \bowtie^T r_2 \rightarrow_M r_2 \bowtie^T r_1$ says that the relations resulting from the left-hand and right-hand sides are equivalent as multisets (and, according to the type ordering, as sets, as well as their snapshots are equivalent as sets and multisets). Meanwhile, the sort push-down rule $sort_A(r_1 \bowtie^T r_2) \rightarrow_L sort_A(r_1) \bowtie^T r_2$, where A belongs to the attribute schema of r_1 and the left-hand side operations are located in the middleware, says that the relations are equivalent as lists and that the other five equivalence types also hold.[1] The latter rule exploits the fact that all temporal join algorithms in the middleware retain the sorting of their left arguments.

Applicability of Transformation Rules. Transformation rules that do not guarantee $\equiv_L$ equivalence cannot always be applied, as illustrated by the following example. Consider a query that performs the above-mentioned temporal join and sorts the result by `Name`. One possible initial plan for this query is shown in Fig. 4(a). The bottom operators represent relations `POSITION` and `SALARY` transferred to the middleware; to achieve this, at least two operations are necessary (a table scan in the DBMS and the actual transfer), but to simplify the example, we view them as one operation and do not consider any transformation rules related to these operations. Temporal join and sorting are performed in the middleware.

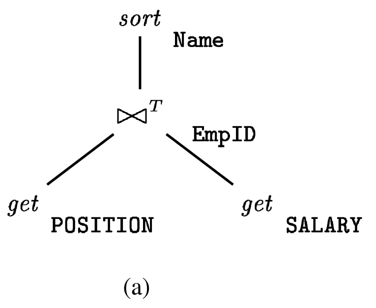
Let us consider rule $r_1 \bowtie^T r_2 \rightarrow_M r_2 \bowtie^T r_1$. This rule can be applied to switch the arguments of the join. However, if we apply the sort push-down rule first and move the sorting below the temporal join, before the temporal join's left argument (leading to the plan shown in Fig. 4(b)), the application of join commutativity rule would lead to an incorrectly ordered query result. Thus, to be able to tell when an $\rightarrow_M$ rule is applicable, the optimizer needs to know the importance of order at each node in the query tree, i.e., whether the result of the operation at the node has to preserve some order or not. In the algebra, this importance is determined by the *OrderRequired* property. To determine the applicability of rules of other types, two additional properties, *DuplicatesRelevant* and *PeriodPreserving*, are used; the first is True if the operation at the node cannot arbitrarily add or remove duplicates, and the second is True if the operation at the node cannot replace its result with a snapshot-equivalent one. For each rule of a given type, Table 1 shows the applicability condition for operator nodes on the left-hand side of the rule.

$$\begin{tikzcd}
L \arrow[d] \arrow[r] & SL \arrow[d] \\
M \arrow[d] \arrow[r] & SM \arrow[d] \\
S \arrow[r] & SS
\end{tikzcd}$$

Fig. 3. Ordering of Equivalence Types

Having an initial query plan, the properties for operators are set in a top-down manner and then adjusted every time a new transformation rule is applied. For the top operator, the properties are set in accordance with the specific user-level query language and query statement, e.g., an SQL query requires the result to be sorted if the `ORDER BY` clause is specified at the outer-most level. Consequently, for the top element, the *OrderRequired* property is set to True only if the `ORDER BY` clause is specified at the outer-most level. The *DuplicatesRelevant* and *PeriodPreserving* properties are always set to True, because we always care about duplicates and time periods. For the

[1] To be precise, the relations are $\equiv_{L,A}$ equivalent, i.e., their projections on A are $\equiv_L$ equivalent. We will use $\equiv_L$ equivalence for simplicity.



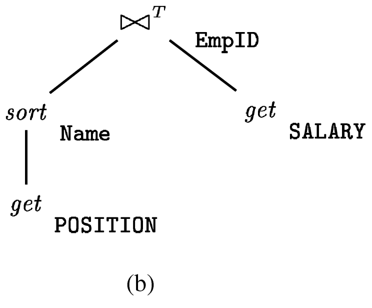


Fig. 4. Query Plans

other operators, the properties are set according to the property values of their parents, e.g., if some operator is the input to the *sort* operator, its *OrderRequired* property will be set to False, because its resulting relation may be replaced (via some transformation rule) by a multiset-equivalent relation, and the correct order of the result will still be ensured by the following *sort* operator. For more details about setting the property values, we refer to [15].

Support in Volcano. Volcano provides a framework of adding new operators and transformation rules, which allows a rather straightforward addition of temporal operators and transfer operators, their cost formulas, selectivity-estimation formulas, and schema propagation formulas. The difficult part is to incorporate different types of transformation rules. While different rule types can be added by just introducing an extra *type* attribute to each rule, to control their applicability is more difficult. The property mechanism cannot directly be used because of Volcano's search-space structure. Having a Volcano search space, values of the three properties cannot be determined for an element, because it is impossible to know the property values of the elements above since the same equivalence-class element may be used as input by different elements of different equivalence classes, as shown later in Fig. 5 where the first element of equivalence class 2 is used both by two elements of equivalence class 3 and by two elements of class 4. Therefore, the determination of the properties can only occur during the actual search, which is performed top down.

Table 1. Applicability of a Rule According to its Type

Rule type	**Applicability condition,** $\forall op \in lhs$
$\rightarrow_L$	True
$\rightarrow_M$	$\neg OrderRequired(op)$
$\rightarrow_S$	$\neg DuplicatesRelevant(op) \wedge \neg OrderRequired(op)$
$\rightarrow^S_L$	$\neg PeriodPreserving(op)$
$\rightarrow^S_M$	$\neg OrderRequired(op) \wedge \neg PeriodPreserving(op)$
$\rightarrow^S_S$	$\neg DuplicatesRelevant(op) \wedge \neg OrderRequired(op) \wedge \neg PeriodPreserving(op)$

3.2 Adjustment of the Search-Space Generation

Since it is impossible to determine properties during the search-space generation, we generate a complete search space by applying transformation rules of *all* types, and then filter away invalid elements during the actual search. The identification of invalid elements is enabled by recording, for each element, a type that represents the combinations of the three property values for which this element may be used. We use six possible type values – L, M, S, SL, SM, SS – which correspond to the six equivalence types. Consequently, the relationship between each element type and the combination of properties corresponds to Table 1. For example, if all properties are True, only L type elements are valid. Intuitively, the element type tells how the relation generated by this element will be equivalent to the first element of the equivalence class.

Figure 5 shows the search space for the query in Fig. 4(a), generated using the join commutativity rule and the sort push-down rule (the first one guarantees $\equiv_M$ equivalence, while the second one guarantees $\equiv_L$ equivalence). Initially, four elements representing the four query-tree operators are created (the first elements of equivalence classes 1–4). Then, the join commutativity rule is applied to the first element of class 3, and a second element representing switched join arguments is added to the class. The sort push-down rule is applied to the first element of class 4, and two new elements are created, one of which is added to class 4 and one of which becomes the only element of class 5. Finally, the join commutativity rule is applied to the second element of class 4, yielding the third element in the class.

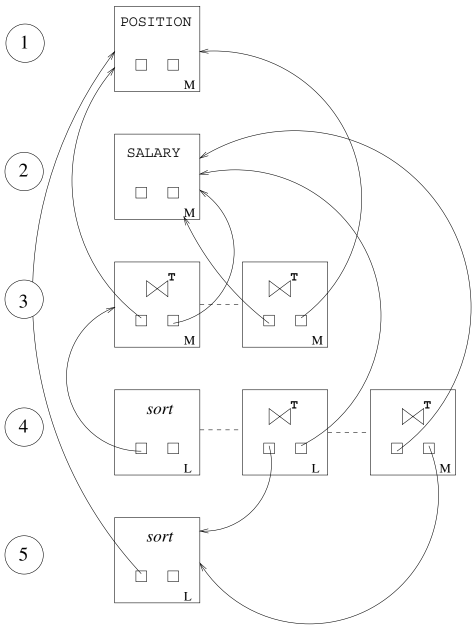


Fig. 5. Search Space

The first elements of classes 1–3 have equivalence type M only, because the base relations are retrieved from the DBMS, and we do not know in which order the DBMS will deliver them. It may happen that a subquery whose top element is the first element of class 3, when run twice, would return relations that are only multiset equivalent.

The third element of class 4 is only $\equiv_M$ equivalent to the other two elements of that class. Since the query requires a sorted result (the *OrderRequired* property value for the top operator is True), only the two first elements of class 4 will be used during plan search. Below, we discuss how the element types are determined.

During the search-space generation, new elements are added after applying transformation rules. For a transformation rule, we give below a procedure for how to set the types of elements resulting from the right-hand side of the rule.

1. The top-element type (the element representing the top operator in the right-hand side of the rule) is set to the type which is the *greatest common descendant* of the transformation-rule type and the types of the elements participating in the left-hand side of the transformation rule.
2. The top-element type is set to a stronger type than specified in 1 only if the right-hand side contains an operation – such as sorting or duplicate elimination – that would enforce a "stronger" equivalence between the new top element and the old top element.
3. The types of other new elements resulting from the right-hand side of the rule are set to any value, but they have to be equal to or stronger than the top-element type.

For example, the greatest common descendant of types M and SM is SS. Let us consider the search space in Fig. 5: the join commutativity rule applied to the second element of class 4 results in the third element of class 4, and its type is set to M, which is the greatest common descendant of L (the type of the second element) and M (the type of the rule).

Now let us consider another query, which performs a selection on relation r transferred to the middleware and then sorts it; see its search space in Fig. 6(a). After transformation rule $sort_A(\sigma_P(r)) \rightarrow_L \sigma_P(sort_A(r))$ is applied to the first element of class 3, the new top element – which becomes the second element of class 3 – is of type L (Fig. 6(b)). Even if the sorting is not at the top level, the result is correctly ordered because the selection retains the order of its argument.

In the given examples, the types of the new non-top elements are set to L. Generally, the types of non-top elements are not important for the correctness, as long as they are not descendants of the new top element type (see the equivalence-type ordering in Fig. 3). Therefore, they should be set aiming to have as small search space as possible, i.e., if an element has to be inserted, first we can look in the existing search space if the same element (with any type) exists there, and if it does, we do not need to insert it anew. If no elements exist, a new element should have L type, because most rules are of $\rightarrow_L$ type and it is likely that, if this element is to be attempted to be inserted again as a top-level element, its type will be L.

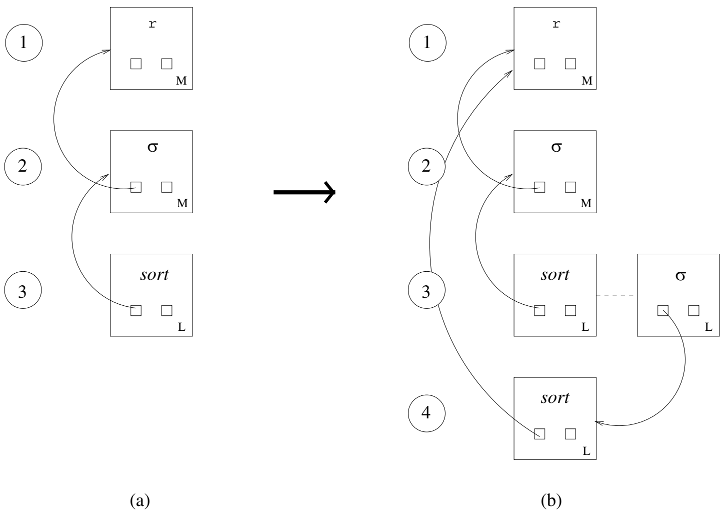


Fig. 6. Search Space Before (a) and After (b) Applying $sort_A(\sigma_P(r)) \rightarrow_L \sigma_P(sort_A(r))$

3.3 Modification of the Plan Search

For the actual search, the code that controls the validity of elements depending on their type has to be added to Volcano.

The most significant change is the addition of properties to the parameter list of the $FindBestPlan$ function. The function uses its input properties to check the validity of its input element, as mentioned in Sect. 3.2, as well as to set the parameter properties for calling itself recursively on the inputs to its input element.

Since equivalence-class elements might be of any of the six different types, each equivalence class may have up to six physical plans, because plans for different-type elements might differ. For example, it is likely that a type M plan will be simpler and less costly than a type L plan. In the $FindBestPlan$ function, when looking if a plan already exists for the input element, we have look for a plan of a type that is stronger than or equal to the input-element type.

4 Experiences

In this section, we consider the extensibility of Volcano in relation to the needs of our framework. We evaluate its support for multiple types of equivalence, discuss other extensions, and evaluate the ease of extensibility.

4.1 Support for Multiple Types of Equivalences

When considering multiple types of equivalences, sorting, duplicate elimination, and coalescing are important operations, because they may change the equivalence type between two relations. For example, if two $\equiv_M$ equivalent relations are sorted on A, their sorted versions will be $\equiv_{L,A}$ equivalent.

Coalescing and duplicate elimination were not implemented in Volcano, and sorting is supported by the so-called *physical properties* of an equivalence class. The possible use of sorting algorithms (termed *enforcers*) is considered during the second phase (plan search) of query optimization. Physical properties are passed as arguments to the $FindBestPlan$ function, and they allow the optimizer to consider different positions of sort enforcers. The use of physical properties increases the code complexity and size – for each algorithm implementing an operator, the optimizer implementor has to write functions deriving physical properties of the algorithm's inputs, checking whether the algorithm satisfies required physical properties, and finding physical properties that are required from the algorithms's inputs.

In our approach, we treat sorting, duplicate elimination, and coalescing as all the other operators and exploit them in the search-space generation, not using physical properties. While it may be possible to pursue a direction where sorting, duplicate elimination, and coalescing are all treated as enforcers and employ physical properties, we feel that this treatment would add unneccesary complexity to the framework because, fundamentally, sorting, coalescing, and duplicate elimination are just like other operators, having their transformation rules and statistics-derivation formulas. Treating them as algorithms reduces the number of transformation rules, but the complexity in the plan-search algorithm is greatly increased. In addition, it would be problematic to incorporate the statistics-estimation formulas for duplicate elimination and coalescing.

4.2 Other Useful Extensions

Our implementation has indicated the need for new or better support in a number of other areas.

The two-stage query optimization of Volcano forced us to apply all types of transformation rules during the first stage. If one stage with a top-down plan search and generation had been used, it would have been easier to control the applicability of the different types of rules and, possibly, would have improved performance.

The search strategy of Volcano is fixed, and no mechanisms for extending or changing it are provided. Proposed improvements of Volcano that were not part of the available code include a mechanism for heuristic guidance, where rules can be ordered according to their "promise" [7]. Such ordering implies that the rules having the best probability to yield better plans would be applied as soon as possible, reducing the overall plan-search time.

We had to add support for equivalence-class elements that point to their own equivalence classes, because this facility was not available in the code supplied. The pointing to the same class often occurs using different equivalence types. For example, sorted relation r is multiset equivalent to r, yielding to a class with two elements (one for r sorted and one for r) where the first (sorting) points to the same class. In addition, we had to implement the linking of classes; the linking is needed when we apply a rule to an element of a certain class and find that the resulting element already exists in some other class, meaning that both classes represent the same logical expression.

The cardinality of a relation resulting from some equivalence class is estimated when the class is created, according to the selectivity estimation method of the operator represented in the first element. When a new element is added to the class, the cardinality is not reestimated. However, the new element may represent an operator for which we may have a better method for estimating the cardinality. For example, it is easier to estimate the size of a join, than the size of a Cartesian product followed by a selection and a projection. Therefore, we had to ensure that the initial plan would contain operators with good cardinality estimation methods.

4.3 Ease of Extensibility

The main challenge for an extensible query optimizer is to balance the efficiency and extensibility, and our study indicates that Volcano's main emphasis is put on the first aspect. Volcano is coded in C and does not follow the object-oriented paradigm, which leads to many interconnected structures, which in turn posed difficulties in figuring out where the structures were defined, initialized, and used. The transformation-rule application code is being generated automatically and does not follow any style guidelines, making it difficult to modify (which was needed when incorporating the necessary modifications in the search-space generation). A lot of arrays and structures have predefined sizes and were not being allocated dynamically, occupying more memory than necessary and providing low scalability. On the other hand, the running times of Volcano (for queries not involving many joins) were quite low, as shown in [16].

The actual implementation tasks, their difficulty, and approximate number of lines of resulting code are summarized in Table 2. We divide the entire implementation effort into three subtasks. The first one, adding support for multiple equivalence types, is the most difficult, and it has been described in the previous sections. Yet the amount of resulting code was rather small. The other task was to add new operators, and while it resulted in a substantial amount of code, it was not difficult, after learning Volcano's provided framework for adding new operators and transformation rules. The same applies to the last task of adding new algorithms; there, however, the amount of code was smaller, because we did not use physical properties.

Support functions form the biggest part of the code added by the optimizer implementor and their size is proportional to the number of operators and algorithms implemented. In our case, we implemented relation retrieval, selection, projection, join, sort-preserving join, temporal join, Cartesian product, duplicate elimination, aggregation, temporal aggregation, and two transfer operators. Similar behavior of many of these operators (particularly, in the propagation of catalog information) resulted in a lot of code repetition in corresponding support functions.

5 Related Work

Our paper takes its outset in the algebraic framework presented in [15]. The framework has been validated by implementing it using the Volcano optimizer and the XXL library of query evaluation algorithms; the architecture, cost and selectivity-estimation formulas, and performance studies have been reported in [16]. The latter paper did not cover the enhancements to Volcano, which are the foci of this paper.

Table 2. Tasks, Their Complexity, and Amount of Code

Task	**Complexity**	**Lines of Code**
Adding equivalence-type support		
Modifying structures	medium	< 200
Modifying search-space generation	high	< 200
Modifying plan search	high	< 200
Adding new operators		
Coding support functions	medium	~ 2500
Coding management of the three properties	medium	~ 400
Coding transformation rules	medium	~ 2300
Adding new algorithms		
Coding support functions	low	~ 1300
Coding implementation rules	medium	< 200

While to our knowledge, nobody has enhanced existing optimizers with support for sets, multisets, and lists, reference [1] reports on experiences from building the query optimizer for Texas Instruments' Open OODB system using Volcano. That paper finds the optimization framework useful, but mentions that much time was spent on writing support and cost functions and that the interface for these tasks is not user-friendly. We agree with these statements, and we draw additional conclusions in Sect, 4.

A number of other extensible query optimizers exist. Volcano evolved from the Exodus optimizer [6], and later was enhanced by the Cascades optimization framework [5], which provides a clean interface and implementation that makes full use of C++ classes, as well as more closely integrates transformation rules and implementation rules, which are distinct sets in Volcano. Since Cascades was intended to be used for Microsoft's SQL Server, its code is not available. Neither is the code for the Starburst query optimizer [8] used in IBM's DB2, nor is the code of the EROC toolkit for building query optimizers [12].

The OPT++ [11] extensible optimizer also uses an object-oriented design with C++ classes to simplify the extension tasks. OPT++ offers a number of search strategies, including "bottom-up" system R-style [14] and the Volcano search strategy; and it can emulate both Starburst and Volcano.

6 Conclusions

A number of extensible query optimizers are available that aim to facilitate changes in query algebras and additions of new functionality. Our study reports on the enhancement of one prominent such extensible query optimizer, Volcano, to support an extended relational algebra, which – in addition to new temporal operators – contains six types of equivalences between relations that lead to six corresponding types of transformation rules. We describe how Volcano's search-space generation and plan search were modified in order to support the algebra, and we evaluate the extensibility of Volcano.

The study indicates that support for sets, multisets, and lists is difficult to add to a pre-existing extensible query optimizer – such support should be considered already during the design of an extensible query optimizer. Volcano's two-staged optimization strategy forces the application of all transformation rules, disregarding their type, during the first stage; if the optimization had occurred in a single stage, we speculate that it would have been easier to control the applicability of rule types and that better performance would have resulted. We also found that, for the modifications we considered, Volcano's interface was not always user-friendly and that the amount of code needed to implement support functions was quite substantial. On the other hand, we found Volcano to be a very useful tool that allowed us to validate our algebra in the middleware architecture more quickly than if we would have had to develop our own optimizer.

This study indicates that extensible query optimizers are useful when testing research ideas and building prototypes. We also believe that extensible optimizers, if developed in industrial strength versions, will prove very useful when building middleware systems that focus on specific functionality suitable for applying conventional relational query optimization techniques. The application of extensible technology to middleware systems is a promising research direction. Due to the increasing use of user-defined routines in conventional DBMSs, optimizer extensibility is also important when creating new DBMSs or modifying existing ones. Finally, the study reported upon here indicates that more research is needed in query optimization and processing that offer integrated support for sets, multisets, and lists.

Acknowledgements. We are grateful to Richard Snodgrass, who took part in the research that lead to the query optimization framework, the Volcano-based implementation of which is reported in this paper.

The research reported here was supported in part by the Wireless Information Management network, funded by the Nordic Academy for Advanced Study through grant 000389, by the Danish Technical Research Council through grant 9700780, and by a grant from the Nykredit Corporation.

References

1. J. A. Blakeley, W. J. McKenna, and G. Graefe. Experiences Building the Open OODB Query Optimizer. In *Proceedings of ACM SIGMOD*, pp. 287–296 (1993).
2. R. Bliujute, S. Saltenis, G. Slivinskas, and C. S. Jensen. Developing a DataBlade for a New Index In *Proceedings of IEEE ICDE*, pp. 314–323 (1999).
3. M. J. Carey and D. Kossmann. Processing Top N and Bottom N Queries. *Data Engineering Bulletin*, 20(3):12–19 (1997).
4. S. Chaudhuri. An Overview of Query Optimization in Relational Systems. In *Proceedings of ACM PODS*, pp. 34–43 (1998).
5. G. Graefe. The Cascades Framework for Query Optimization. *Data Engineering Bulletin*, 18(3):19–29 (1995).
6. G. Graefe and D. J. DeWitt. The Exodus Optimizer Generator. In *Proceedings of ACM SIGMOD*, pp. 160–172 (1987).
7. G. Graefe and W. J. McKenna. The Volcano Optimizer Generator: Extensibility and Efficient Search. In *Proceedings of IEEE ICDE*, pp. 209–218 (1993).
8. L. M. Haas et al. Starburst Mid-Flight: As the Dust Clears. *IEEE TKDE*, 2(1):143–160 (1990).
9. Informix Software. DataBlade Overview. URL: <`www.informix.com/products/options/udo/datablade/`>, current as of May 29, 2001.

10. M. Jaedicke and B. Mitschang. User-Defined Table Operators: Enhancing Extensibility for ORDBMS. In *Proceedings of VLDB*, pp. 494-505 (1999).
11. N. Kabra and D. J. DeWitt. OPT++: An Object-Oriented Implementation for Extensible Database Query Optimization. *VLDB Journal*, 8(1):55–78 (1999).
12. W. J. McKenna, L. Burger, C. Hoang, and M. Truong. EROC: A Toolkit for Building NEATO Query Optimizers. In *Proceedings of VLDB*, pp. 111–121 (1996).
13. Oracle Technology Network. Overview of PL/SQL. URL: <`otn.oracle.com/tech/pl_sql/`>, current as of May 29, 2001.
14. P. G. Selinger et al. Access Path Selection in a Relational Database Management System. In *Proceedings of ACM SIGMOD*, pp. 23–34 (1979).
15. G. Slivinskas, C. S. Jensen, and R. T. Snodgrass. A Foundation for Conventional and Temporal Query Optimization Addressing Duplicates and Ordering. *IEEE TKDE*, 13(1):21–49 (2001).
16. G. Slivinskas, C. S. Jensen, and R. T. Snodgrass. Adaptable Query Optimization and Evaluation in Temporal Middleware. In *Proceedings of ACM SIGMOD*, pp. 127–138 (2001).

Information Sources Registration at a Subject Mediator as Compositional Development

Dmitry O. Briukhov, Leonid A. Kalinichenko, and Nikolay A. Skvortsov

Institute for Problems of Informatics RAS
{brd,leonidk,scvora}@synth.ipi.ac.ru

Abstract. Method for heterogeneous information source registration at subject mediators with local as view (LAV) organization is presented. LAV approach considers schemas exported by sources as materialized views over virtual classes of the mediator. This approach is intended to cope with a dynamic, possibly incomplete set of sources. To disseminate the information sources, their providers should register them at a respective subject mediator. Such registration can be done concurrently and at any time.
The registration method proposed is new and contributes to the following. The method is applicable to wide class of source specification models representable in hybrid semistructured/object canonical mediator model. Ontological specifications are used for identification of mediator classes semantically relevant to a source class. Maximal subset of source information relevant to the mediator classes is identified. Concretizing types are defined so that federated classes instance types are refined by the source instance type. This direction naturally supports query planning refining a mediator query in terms of a specific source. Such refining direction is in contrast to conventional compositional development where specification of requirements is to be refined by specifications of components. Such inversion is natural for the registration process: a materialized view (requirements) is constructed over virtual specifications (components).

1 Introduction

Mediation of heterogeneous information sources provides an approach for intelligent information integration. Mediation architecture introduced in [20] defines an idea of a middleware positioned between information sources (information providers) and information consumers. Mediators support modelling facilities and methods for conversion of unorganized, nonsystematic population of autonomous information sources keeped by different information providers into a well-structured information collection defined by the integrated uniform specifications. Mediators provide also a uniform query interface to the multiple data sources, thereby freeing the user from having to locate the relevant sources, query each one in isolation, and combine manually the information from them. Important application areas greatly benefit from the *subject mediation* approach supporting information integration in a particular subject domain. Among them are Web information integration systems, digital libraries providing content interoperability, digital repositories of knowledge in certain domains (like: Digital Earth, Digital Sky, Digital Bio, Digital Law, Digital Art, Digital Music).

For such areas, according to the approach, the application domain model is to be defined by the experts in the field independently of relevant information sources. This model may include specifications of data structures, terminologies (thesauri), concepts (ontologies), methods applicable to data, processes (workflows), characteristic for the domain. These definitions constitute specification of a subject mediator. After subject mediator had been specified, information providers can disseminate their information for integration in the subject domain independently of each other and at any time. To disseminate they should register their information at the subject mediator. Users may not know anything about the registration process and about the sources that have been registered. Users should know only subject domain definitions that contain concepts, structures, methods approved by the subject domain community. Thus various information sources belonging to different providers can be registered at a mediator.

The subject mediation approach is applicable to various subject domains in science, cultural heritage, mass media, e-commerce, etc. This technology is considered as a promising alternative to the widely used general purpose Web search engines characterized by very low precision of search due to uncontrollable use of terms for indexing and search. This is unavoidable payment for simplicity of sites "registration" at the engines.

Two basic approaches are known for the mediator architectures [6]. According to the first one, called Global as View (GAV), the global schema is constructed by several layers of views above the schemas exported by pre-selected sources. Queries are expressed in terms of the global schemas and are evaluated similarly to the conventional federated database approaches [17]. TSIMMIS [7] or HERMES [18] apply this architecture. Another approach, known as Local as View (LAV), considers schemas exported by sources as materialized views over virtual classes of the mediated schema. Queries are expressed in terms of the mediated schema. Query evaluation is done by query planning making its rewriting in terms of the source schemas. Information Manifold [14] or Infomaster [5] apply this strategy. The LAV architecture is designed to cope with a dynamic, possibly incomplete set of sources. Sources may change their exported schemas, become unavailable from time to time. LAV is potentially scalable with respect to a number of sources involved. Further in this paper we assume LAV as an approach suitable for the subject mediation.

Two separate phases of the subject mediator's functioning are distinguished: *consolidation* phase and *operational* phase. The consolidation phase is intended for the subject model definition. During this phase the mediator's federated schema metainformation is formed. The technique used for that is beyond this paper.

During the operational phase the burden of the sources registration process is imposed on the information providers. They formulate sources' specifications (schemas, concept definitions, vocabularies) in terms of the subject mediator's federated metainformation and develop the required wrappers. In such process of registration the local metainformation sublayer of the mediator is formed expressing local schemas in the mediator's canonical model as views above the federated schema. Making source registration concurrently by providers is the way to reach the mediator's scalability.

During the registration a local source class is modelled as a set of instances (objects) of the class instance type, and the description of the source in terms of the federated schema specifies the constraints on the class instances to be admissible for the subject mediator. Formally, the content of a local source class is described by a formulae simplified as $C(z) \subseteq \exists \bar{x}(C_1(\bar{x_1})$ &...& $C_n(\bar{x_n})$ & $Con)$ where C is a local source class, C_1, ..., C_n are federated schema classes, z is a reduct [10] of the local class instance type being a concretization of a reduct of resulting instance type of the conjunctive formula that includes only those attributes of the resulting type whose names are not bound by the existential quantifier, Con is additional constraints imposed by formulae. That is, the defined reduct of an instance obtained from the local source class should satisfy the constraint expressed by the formula. Of course, this description does not imply that the local source contains all the instances that satisfy the formula. Such representation of local sources means that we do not have to add a local source class to the federated level whenever sources are added, since this class does not have to correspond directly to a federated class.

General idea of representation of the local classes in terms of federated classes is similar to the one proposed in [14]. Main differences of the current approach consist in taking into account issues more relevant to real environments, such as using general type model and type specification calculus [10], applying the refining mapping of the local specific data models into the canonical model of the mediator [12], resolving ontological differences between federated and local concepts [2], systematic resolving structural, behavioral and value conflicts of local and federated types and classes [2].

This paper focuses on analysis methods and tools required to support information source registration process at the mediator. Main contribution of the paper is that for the LAV mediation strategy the paper considers the information source registration as the process of compositional information systems development [2]. Local source metainformation definitions are treated as specifications of requirements and classes of the federated level with the related metainformation – as specifications of pre-existing components. To get local classes definitions as views above the federated level with constraints given in the form shown above, the facilities of the modified compositional information systems development method and tool [2] are applied. This modified method and tool are considered to be a part of the mediator's metainformation support. The method considered can be useful also for a Data Warehouse (DW) environment. Main difference between the mediator's and DW approaches consists in that a mediator schema is virtual though DW schema is materialized.

At the moment of the paper writing no works considering a source registration at a LAV-approached mediator as a design process have been identified. Related papers in DW design could have also been expected. Analysis of the latter is given in the Related Work section. The analysis shows that DW design have different motivation and apply different methods and models comparing to those that are needed for a source registration at the LAV-approached mediators.

The paper is composed as follows. After brief characterization of the canonical model and metainformation support at a mediator, the registration process is treated in detail. The process of compositional development leading to the definition of source classes in terms of classes of the federated level is introduced. Contextualization of a source at the federated level is a part of this process. Various aspects of the registration process are illustrated by an example. Cultural heritage is taken as an application domain assuming that a respective mediator has been defined. For the sources we exploit slightly modified schemas of Web sites of Louvre and Uffizi museums and Z39.50 CIMI profile. Due to the paper size limitation, only some classes of the Uffizi site are included.

2 Related Works

In [4] the DW Schema is considered to be a description of logical content of the materialized views constituting the DW. Each portion of such schema is described in terms of a set of definitions of relations, each one expressed in terms of a query over the DW Domain Model. A view is actually materialized starting from the data in the sources by means of software components, called mediators. Source is defined both on a logical level (relations) and on a conceptual level (E/R). A query is provided to express logical source as a view above conceptual definitions. Integration is seen as the incremental process of understanding and representing the relationships between data in the sources, rather than simply producing a unified data schema. Description logic which treats n-ary relations as first-class citizens is used for the conceptual modeling of both the subject domain and the various sources. The approach [4] resembles a consolidation phase of a GAV mediator.

A common understanding of a "well-designed" DW schema in the literature [9] is that such schema should have the form of a "star", i.e., it should consist of a central fact table that contains the facts of interest to an application, and that is connected to a number of dimension tables through referential integrity constraints based on the various dimension keys. Since dimensions can be composed of attribute hierarchies, it is often the case that dimension tables are unnormalized, and their normalization results in what is known as a snowflake schema.

Known research works in DW design (including [4,8,9,19]) have different motivation and apply different methods and models comparing to those that are needed for a source registration at the LAV-approached mediators or at DW with a predefined conceptual schema.

3 Fundamentals of the Compositional IS Development Method

The main distinguishing feature of the method considered is a creation of compositions of component specification fragments refining specifications of requirements. Widely used component-based development methods (e.g., JavaBeans) construct aggregates of components differently – just linking ports of components with each other or considering their interactions on the contractual basis [15].

Refining specifications obtained during the compositional development, according to the refinement theory, can be used anywhere instead of the refined specifications of requirements without noticing such substitutions by the users. The refinement methods [1] allow to justify the fact of a refinement formally to guarantee the adequacy of the specifications obtained to that of the required.

3.1 Compositional Specification Calculus

The design is a process of systematic manipulation and transformation of specifications. Type specifications of the canonical model (SYNTHESIS) [13] are chosen as the basic units for such manipulation. The manipulations required include decomposition of type specifications into consistent fragments, identification of reusable fragments (patterns of reuse), composition of identified fragments into specifications concretizing the requirements, justification of reusability and substitutability of the results of such transformations instead of the specifications of requirements. The compositional specification calculus [10] intentionally designed for such manipulations uses the following concepts and operations.

A signature Σ_T of a type specification $T =< V_T, O_T, I_T >$ includes a set of operation symbols O_T indicating operations argument and result types and a set of predicate symbols I_T (for the type invariants) indicating predicate argument types. Conjunction of all invariants in I_T constitutes the type invariant. We model an extension V_T of each type T (a carrier of the type) by a set of proxies representing respective instances of the type.

Definition 1. Type reduct *A signature reduct R_T of a type T is defined as a subsignature Σ'_T of type signature Σ_T that includes a carrier V_T, a set of symbols of operations $O'_T \subseteq O_T$, a set of symbols of invariants $I'_T \subseteq I_T$.*

This definition from the signature level can be easily extended to the specification level so that a type reduct R_T can be considered a *subspecification* (with a signature Σ'_T) of specification of the type T. The specification of R_T should be formed so that R_T becomes a supertype of T. We assume that only the states admissible for a type remain to be admissible for a reduct of the type (no other reduct states are admissible). Therefore, the carrier of a reduct is assumed to be equal to the carrier of its type.

Definition 2. *Type U is a* refinement *of type T iff*

- *there exists a one-to-one correspondence* $Ops : O_T \Leftrightarrow O_U$;
- *there exists an abstraction function* $Abs : V_T \rightarrow V_U$ *that maps each admissible state of* T *into the respective state of* U;
- $\forall x \in V_T\ \exists y \in V_U (Abs(x, y) \Rightarrow I_T \wedge I_U)$
- *for every operation* $o \in O_T$ *the operation* $Ops(o) = o' \in O_U$ *is a refinement of* o. *To establish an operation refinement it is required that operation precondition* $pre(o)$ *should* imply *the precondition* $pre(o')$ *and operation postcondition* $post(o')$ *should* imply *postcondition* $post(o)$.

Based on the notions of reduct and type refinement, a measure of common information between types in $\mathcal{T}$ can be established.

Definition 3. *A* common reduct *for types* T_1, T_2 *is such reduct* R_{T_1} *of* T_1 *that there exists a reduct* R_{T_2} *of* T_2 *such that* R_{T_2} *is a refinement of* R_{T_1}. *Further we refer to* R_{T_2} *as to a* conjugate *of the common reduct.*

Definition 4. *A* most common reduct $R_{MC}(T_1, T_2)$ *for types* T_1, T_2 *is a reduct* R_{T_1} *of* T_1 *such that there exists a reduct* R_{T_2} *of* T_2 *that refines* R_{T_1} *and there can be no other reduct* $R^i_{T_1}$ *such that* $R_{MC}(T_1, T_2)$ *is a reduct of* $R^i_{T_1}$, $R^i_{T_1}$ *is not equal to* $R_{MC}(T_1, T_2)$ *and there exists a reduct* $R^i_{T_2}$ *of* T_2 *that refines* $R^i_{T_1}$.

Reducts provide for type specification decompositions thus creating a basis for their further compositions. Type composition operations can be used to infer new types from the existing ones. We introduce here definition of only one operation - *join*. Let $T_i (1 \leq i \leq n) \in \mathcal{T}$ denotes types.

Definition 5. Type join *operation. An operation* $T_1 \sqcup T_2$ *produces type* T *as a 'join' of specifications of the operand types. Generally* T *includes a merge of specifications of* T_1 *and* T_2. *Common elements of specifications of* T_1 *and* T_2 *are included into the merge (resulting type) only once. The common elements are determined by another merge - the merge of conjugates of two most common reducts of types* T_1 *and* T_2 : $R_{MC}(T_1, T_2)$ *and* $R_{MC}(T_2, T_1)$. *The merge of two conjugates includes union of sets of their operation specifications. If in the union we get a pair of operations that are in a refinement order then only one of them, the more refined one (belonging to the conjugate of the most common reduct) is included into the merge. Invariants created in the resulting type are formed by* conjuncting *invariants taken from the original types.*

A type T *is placed in the type hierarchy as an immediate subtype of the join operand types and a direct supertype of all the common direct subtypes of the join argument types.*

Operations of the compositional calculus form a *type lattice* [10] on the basis of a subtype relation (as a partial order).

3.2 Design Phase

Design is the component-based process of concretization of a specification obtained on an analysis phase by an interoperable composition of pre-existing information components. It includes reconciliation of application domain (here, federated schema) and of information source ontological contexts establishing the ontological relevance of constituents of requirements and components specifications, identification of component types (classes) and their fragments (identifying most common reducts) suitable for the concretization of an analysis model type (class) capturing its structural, extensional and behavioural properties, composition (using type operations *join* and *meet*) of such fragments into specifications concretizing the requirements, justification of a property of refinement of requirements by such compositions.

4 Registration of Information Sources at a Subject Mediator

4.1 General Schema of a Process of an Information Source Registration at the Mediator

As a preliminary step, provide an ontological integration of an information source specification with the federated level specification. The process of the ontological integration is the same as for the compositional development method.

After the ontological integration, for each *source* class the following steps are required:

1. **relevant federated classes identification**
   Find federated classes that ontologically can be used for defining source class extent in terms of federated classes. To a source class several federated classes may correspond covering with their instance types different reducts of an instance type of the source class. On another hand, several source classes may correspond to one federated class.
2. **most common reducts construction**
   For an instance type of each identified *federated* class do:
   a) Construct most common reducts for instance type of this federated class and source class instance type to concretize (partially) such federated instance type. Most common reduct may include also additional attributes corresponding to those federated type attributes that can be derived from the source type instances to support them.
   b) In this process for each attribute type of the common reduct a concretizing type, concretizing function or their combination should be constructed (this step should be recursively applied).
3. **partial source view construction**
   a) For each relevant federated class construct a partial source view expressing a constraints in terms of the federated class that should be satisfied by values of respective most common reducts of source class instances. Thus partial views over all relevant federated classes will be obtained.
4. **partial views composition**
   a) Construct compositions of the source type most common reducts obtained for instance types of all federated classes involved.

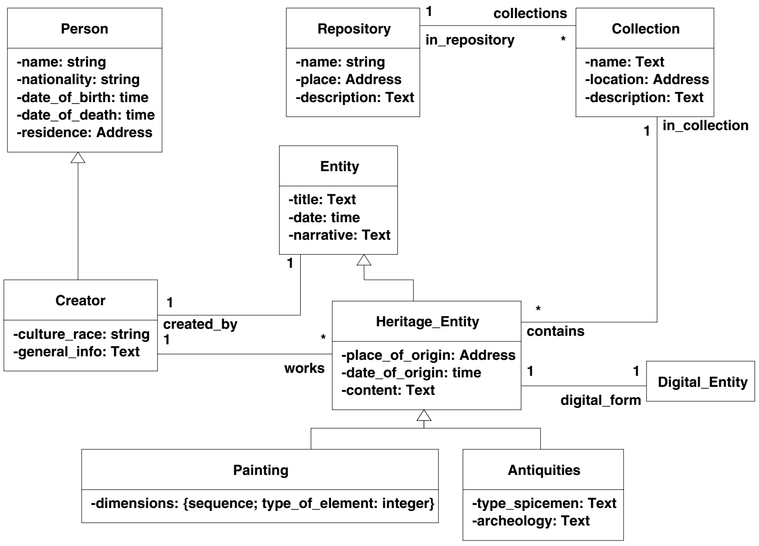


Fig. 1. Specifications of types of the federated schema

b) Construct a source view as a composition of partial views obtained above. This is an expression of a materialized view of an information source in terms of federated classes. An instance type of this view is determined by the most common reducts composition constructed above.

4.2 Example of Subject Mediator and Information Source Specifications

During the rest of the paper we demonstrate our approach on an example showing the registration of the schema of the Uffizi museum Web site at a subject mediator. Cultural heritage is taken as an application domain of the mediator. For the federated and local source type schema we use UML diagrams and do not provide canonical model specifications to save space.

Figure 1 shows a part of the federated schema of the mediator for the cultural heritage domain.

Figure 2 shows a part of the schema of the Uffizi museum Web site. We apply its description similar to one used in the Araneus project.

Text and *Textual* types define interfaces of different types used for textual search (in Z39.50 and in Oracle respectively). Establishing of a refinement relationships for these type is beyond this paper.

4.3 Ontological Integration Example

Ontological integration is used to establishing ontological relevance of constituents of mediator's federated schema and local source schema specifications. Spec-

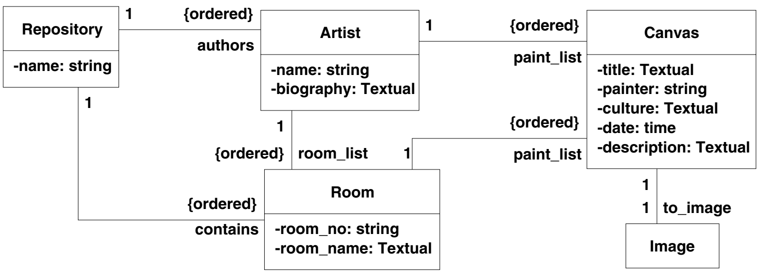


Fig. 2. Specifications of types of the Uffizi site schema

ifications of federated schema and specifications of local sources must be associated with ontological contexts containing concepts of the respective subject areas. Each element of specifications must be associated with an ontlogical concept, describing the element. Elements (types, classes, attributes and others) are defined as instances of semantically relevant ontological classes (concepts).

The ontological concepts have their verbal definitions and descriptor lists. Verbal concept definitions are similar to definitions of words in an explanatory dictionary. Descriptor lists included in the specifications of concepts are built on the basis of the meaningful words lists in the concept's verbal definition. Concept descriptors are required for establishing relationships between concepts. Hypernym/hyponym relationships and positive relationships (synonym) can be defined between ontological concepts. These relationships can be treated as fuzzy ones.

Relationships between concepts of different contexts are established by calculating the correlation coefficients between concepts on the basis of their verbal definitions. The correlation coefficients are calculated using the vector-space approach [16,3]:

$$sim(X,Y) = \frac{\sum_{k=1}^{t}(W_{Xk} \cdot W_{Yk})}{\sqrt{\sum_{k=1}^{t}(W_{Xk})^2 \cdot \sum_{k=1}^{t}(W_{Yk})^2}}$$

The range of values of the simulating function $sim(X,Y)$ is the real interval [0.0,1.0]. The concept X is considered positively related to the concept Y if $sim(X,Y)$ is greater than a certain threshold value ℓ; in this case, the intercontext positive relationship is established between X and Y, and the value of the function is considered as the relationship strength.

Let's consider the federated concept *IconographicObject* which associated with the element *Painting* of federated schema and the local concept *ArtWork* which associated with the element *Canvas* of local source schema. Specifications of the concepts look as follows:

```
{IconographicObject;
  in: concept;
  supertype: ManMadeObject, ConceptualObject;
  def: "This entity comprises objects which are designed primarily or in
  addition to another functionality to represent or depict something in
  an optical manner, be it concrete or abstract. This entity has a
  certain pragmatic value in the fine arts since it conveniently groups
  together objects such as paintings, drawings, watercolours and other
  similar objects."
}

{ArtWork;
  in: concept;
  def: "Art such as sculpture, drawings or paintings depicting something"
}
```

The correlation coefficient of the similarity function is the following:

```
sim(IconographicObject, ArtWork) = 0.2491
```

Basing on this coefficient we establish the positive relationship between these concepts. The correlation between the schema elements *Painting* and *Canvas* is established with the same strength. Analogously hypernym/hyponym concept relationships are established [3].

4.4 Most Common Reducts Construction

Construct most common reducts for the federated types *Painting* and *Creator* and their ontologically relevant source type *Canvas* (Figs. 1, 2). On construction of the most common reduct of types defined in the specification of requirements (local source definition) and in a component (federated schema classes), various conflicts between the specifications should be discovered and resolved. The result of the conflicts resolution is represented in a transformation of a conjugate reduct into a concretizing reduct. The concretizing reduct specification includes together with the attributes of a reduced type the mapping of a concretizing reduct attributes into attributes of a common reduct and the conflict resolution functions.

The most common reducts and their concretizing reducts are specified as SYNTHESIS types. Specific metaslots and attributes are used to represent specific semantics. Specification of common reduct for *Painting* and *Canvas* denoted as *R_Painting_Canvas* is defined as follows:

```
{R_Painting_Canvas;
  in: reduct;
  metaslot
    of: Painting;
    taking: {title, created_by, date_of_origin, narrative, digital_form,
             in_collection};
    c_reduct: CR_Painting_Canvas;
  end;
}
```

A slot *of* refers to the reduced federated type. A list of attributes of the reduced type in the slot *taking* contains names of its attributes that are to be included into the common reduct.

A slot *c_reduct* refers to the concretizing reduct based on a source type. We add a concretizing reduct for the *Painting*, *Canvas* pair of types as the following type definition:

```
{CR_Painting_Canvas;
  in: c_reduct;
  metaslot
      of: Canvas;
      taking: {title, painter, date, description, to_image};
      reduct: R_Painting_Canvas
  end;
  simulating: {
    R_Painting_Canvas.title ~ CR_Painting_Canvas.title;
    R_Painting_Canvas.created_by ~ CR_Painting_Canvas.get_created_by;
    R_Painting_Canvas.date_of_origin ~ CR_Painting_Canvas.date;
    R_Painting_Canvas.narrative ~ CR_Painting_Canvas.description;
    R_Painting_Canvas.digital_form ~ CR_Painting_Canvas.to_image;
    R_Painting_Canvas.in_collection ~ CR_Painting_Canvas.get_in_collection
  };
  get_created_by: {in: function;
    params: {+ext/CR_Painting_Canvas, -returns/Creator};
    predicative: {ex c/Canvas ((c/CR_Painting_Canvas = ext) &
      ex a/Artist ((c.painter=a.name) & returns=a/CR_Creator_Artist))}
  };
  get_in_collection: {in: function;
    params: {+ext/CR_Painting_Canvas, -returns/Collection}
    predicative: {ex c/Canvas ((c/CR_Painting_Canvas = ext) &
      ex r/Room ((in(c,r.paint_list) & returns=r/CR_Room_Collection))}
  }
}
```

A slot *of* refers to the source type. In this case a slot *reduct* refers to the respective common reduct. The predicate *simulating* shows how the concretizing state is mapped into the common reduct state.

E.g., *R_Painting_Canvas.date_of_origin* $\sim$ *CR_Painting_Canvas.date* defines that attribute *date_of_origin* of reduct *R_Painting_Canvas* is refined by attribute *date* of concretizing reduct *CR_Painting_Canvas* and values of attribute *date_of_origin* are taken from values of attribute *date*. *get_created_by* presents the mediating function resolving the conflict in mixed pre- and post-conditions.

The most common reduct and concretizing reduct for types *Creator* and *Canvas* are constructed similarly.

4.5 Source View Construction

Now for each relevant federated class we construct a formula expressing a constraint in terms of the federated classes that should be satisfied by local class instances.

The formula expressing the local class *canvas* is terms of the federated class *painting* is defined as:

$$canvas(p/CR\_Painting\_Canvas) \subseteq painting(p/R\_Painting\_Canvas)\ \&\ p.in\_collection.in\_repository =' Uffizi'$$

Specification of a class (actually, this is *local as view* class) containing this formula is:

```
{v_canvas_painting;
  in: class;
  class_section: {
    key: invariant, {unique; {title}};
    lav: invariant, {subseteq (v_canvas_painting(p),
      painting(p/R_Painting_Canvas) &
      p.in_collection.in_repository = 'Uffizi')}
  };
  instance_section: CR_Painting_Canvas
}
```

Now, another formula expressing the local class *canvas* is terms of the federated class *creator* is defined as:

$$canvas(c/CR\_Creator\_Canvas) \subseteq creator(c/R\_Creator\_Canvas)\ \&\ \exists w/Painting(in(w, c.works)\ \&\ w.in\_collection.in\_repository =' Uffizi')$$

Specification of a respective class *v_canvas_creator* denoting local source as view is similar to the specification of class *v_canvas_painting*.

Now we apply type compositions and identify a useful part of *Canvas* type as a composition of type reducts obtained before. We consider a concretizing reduct *CR_Painting_Creator_Canvas* as the join of the concretizing reducts *CR_Painting_Canvas* and *CR_Creator_Canvas*.

$$CR\_Painting\_Creator\_Canvas = CR\_Painting\_Canvas \sqcup CR\_Creator\_Canvas$$

A final formula for a local class *canvas* in terms of the federated classes *painting* and *creator* is created as a conjunction of the partial constraints:

$$canvas(p/CR\_Painting\_Creator\_Canvas) \subseteq painting(p/R\_Painting\_Canvas)\ \&\ p.in\_collection.in\_repository =' Uffizi'\ \&\ creator(c/R\_Creator\_Canvas)\ \&\ \exists w/Painting(in(w, c.works)\ \&\ w.in\_collection.in\_repository =' Uffizi')$$

Complete definition of source view looks as follows:

```
{v_canvas;
  in: class;
  class_section: {
    key: invariant, {unique; {title}};
    lav: invariant, {subseteq(v_canvas,
       painting(p/R_Painting_Canvas) &
       p.in_collection.in_repository = 'Uffizi' &
       creator(c/R_Creator_Canvas) & ex w/Painting (in(w,c.works) &
       w.in_collection.in_repository = 'Uffizi')})
  };
  instance_section: CR_Painting_Creator_Canvas;
}
```

The last composition may be considered as a *join* composition of types defined in class sections of partial views (that is, as a composition of types of classes treated as objects).

5 Conclusion

The paper presents a method for heterogeneous information sources registration at subject mediators. Source specifications as materialized views above virtual classes of mediator are designed applying compositional development method (source definition is treated as a specification of requirements and class definitions of federated schema are treated as component specifications). This approach is intended to cope with a dynamic, possibly incomplete set of sources. Sources may change their exported schemas, become unavailable from time to time. To disseminate the information sources, their providers should register them at a respective subject mediator. Such registration can be done concurrently and at any time. To make subject mediators scalable with respect to a number of sources involved, specific methods and tools supporting process of information sources registration are required. The method is applicable to wide class of source specification models representable in hybrid semistructured/object canonical mediator model [11].

The registration tool is being developed reusing the SYNTHESIS compositional design method prototype [2]. The tool is based on Oracle 8i and Java 2 under Windows environment.

References

1. R.-J. Back, J. von Wright. Refinement Calculus: A systematic Introduction. Springer Verlag, 1998
2. D. O. Briukhov, L. A. Kalinichenko. Component-Based Information Systems Development Tool Supporting the SYNTHESIS Design Method. In *Proc. of the East European Symposium on "Advances in Databases and Information Systems"*, Poland, Springer, LNCS No.1475, 1998

3. D. O. Briukhov, S. S. Shumilov. Ontology Specification and Integration Facilities in a Semantic Interoperation Framework, In *Proc. of the International Workshop ADBIS'95*, Springer, 1995
4. D. Calvanese, G. De Giacomo, M. Lenzerini, D. Nardi, R. Rosati. Source Integration in Data Warehousing, In *Proc. of the 9th Int. Workshop on Database and Expert Systems Applications (DEXA-98)*, pages 192-197. IEEE Computer Society Press, 1998
5. O. Duschka and M. Genesereth. Answering Queries Using Recursive Views. In *Principles Of Database Systems (PODS)*, 1997
6. M. Friedman, A. Levy, and T. Millstein. Navigational Plans for Data Integration, 1999
7. H. Garcia-Molina, J. Hammer, K. Ireland, Y. Papakonstantinou, J. Ullman, and J. Widom. The Tsimmis Approach to Mediation: Data Models and Languages. Journal of Intelligent Information System, 1997
8. M. Golfarelli, D. Maio, S. Rizzi. Conceptual Design of Data Warehouses from E/R Schemes In *Proc. of the 31st Hawaii International Conference on System Sciences*, Kona, Hawaii, 1998
9. B. Husemann, J. Lechtenborger, G. Vossen. Conceptual Data Warehouse Design. In *Proc. of the International Workshop on Design and Management of Data Warehouses (DMDW'2000)*, Stockholm, Sweden, June 5-6, 2000
10. L. A. Kalinichenko. Compositional Specification Calculus for Information Systems Development. In *Proc. of the East-West Symposium on Advances in Databases and Information Systems (ADBIS'99)*, Maribor, Slovenia, September 1999, Springer Verlag, LNCS, 1999
11. L. A. Kalinichenko. Integration of heterogeneous semistructured data models in the canonical one. In *Proc. of First Russian National Conference on "Digital Libraries: Advanced Methods and Technologies, Digital Collections"*, Saint-Petersburg, October 1999
12. L. A. Kalinichenko. Method for data models integration in the common paradigm. In *Proc. of the First East European Workshop 'Advances in Databases and Information Systems'*, St. Petersburg, September 1997
13. L. A. Kalinichenko. SYNTHESIS: the language for desription, design and programming of the heterogeneous interoperable information resource environment. Institute for Problems of Informatics, Russian Academy of Sciences, Moscow, 1995
14. A. Levy, A. Rajaraman, and J. Ordille. Querying Heterogeneous Information Sources using Source Descriptions. In *Proc. of the 22nd Conf. on Very Large Databases*, pages 251-262, 1996
15. M. Lumpe. A Pi-Calculus Based Approach to Software Composition, Ph.D. thesis, University of Bern, Institute of Computer Science and Applied Mathematics, January 1999
16. G. Salton, C. Buckley. Term-Weighting Approaches in Automatic Text Retrieval. Readings in Information Retrieval, K. S. Jones and P. Willett, Kaufmann, 1997
17. A. Sheth and J. Larson. Federated Database Systems for Managing Distributed, Heterogeneous, and Autonomous Database. ACM Computing Surveys, 1990
18. V. S. Subrahmanian. Hermes: a Heterogeneous Reasoning and Mediator System. `http://www.cs.umd.edu//projects/hermes/publications/postscripts/tois.ps`
19. N. Tryfona, F. Busborg, J. G. Borch Christiansen. starER: A Conceptual Model for Data Warehouse Design, ACM Second International Workshop on Data Warehousing and OLAP (DOLAP), 1999
20. G. Wiederhold. Mediators in the Architecture of Future Information Systems. IEEE Computer, 1992

Extracting Theme Melodies by Using a Graphical Clustering Algorithm for Content-Based Music Information Retrieval

Yong-Kyoon Kang, Kyong-I Ku, and Yoo-Sung Kim

Department of Computer Science & Engineering
INHA University, INCHEON 402-751, Korea
yskim@inha.ac.kr

Abstract. We proposed the mechanism of extracting theme melodies from a song by using a graphical clustering algorithm. In the proposed mechanism, a song is split into the set of motifs each of which is the minimum meaningful unit. Then the system clusters the motifs into groups based on the similarity values calculated between all pairs of motifs so that each cluster has higher similarity values between them than others. From each clusters, the system selects a theme melody based on the positions of the motif within a song and the maximum summation of similarity values of edges adjacent to the motif node in each cluster. As the experimental results, we showed an example in which we describe how the theme melodies of a song can be extracted by using the proposed algorithm.

1 Introduction

As user's requests for systematic management of multimedia information have been increased continuously, the content-based multimedia information retrieval is necessary to satisfy user's requirement. For music information, the traditional music retrieval mechanisms using the metadata of music have the major restriction; users should remember some appropriate metadata of a song to make the retrieval query. If users do not know the metadata of the song they want, they might not retrieve the song. This problem is caused by the general fact that people prefer to remember a part of song rather than its metadata. To solve the restriction of the traditional music retrieval mechanism using the metadata, the effective content-based music retrieval mechanism is needed. In content-based music retrieval system, users input some part of a song they remembering then the system retrieves songs that contain some variations of the given melodies.

As the content-based music retrieval systems, several systems have been developed ([1,2]). However, the previous systems have two major problems. First, these systems do not have indexing mechanism that is helpful to improve the retrieval performance.

Second these systems do not use the full primitive features that are essential to represent the semantics of music.

To solve the first problem, previous researches([3,4,5]) have proposed theme melody index schemes in which the theme melodies extracted from a song are included. However, in [3], [4], and [5], theme melodies are represented by using only the pitches of notes within a melody. Furthermore, they consider the exactly repeated patterns within a music object as the theme melodies of the music. That is, the previous extracting mechanism can not extract the approximately(not exactly) repeated patterns as the theme melodies. In general, however, theme melodies can be repeated more than once with some variations within a song. Hence, an extraction mechanism that must deal with some variation of theme melody is highly required for the effective content-based music retrieval.

In our previous research([6]), we proposed a similarity computation algorithm between music objects based on their contents, not only pitches but also durations(lengths) of notes. Also, in our another previous research([7]), we showed how the theme melodies can be used as the indexing terms for the efficient retrieval. However, we did not fully describe how the approximately repeated theme melodies are extracted from a song.

In this paper, we proposed a theme melody extraction mechanism in which a modified version of the graphical clustering algorithm of [12] is used for grouping the approximately repeated motifs into a cluster. In the first step of the extraction mechanism, we split a music file into the set of motifs each of which is the minimum meaningful unit. Then the system extracts pitches and lengths of notes as the song's primitive. Then the system calculates the similarity values between all pairs of motifs within a song based on the pitches and lengths of the corresponding notes between two motifs and clusters the motifs into groups based on the similarity values so that each cluster has higher similarity values between them than others. From each cluster, the system selects theme melody based on the positions of the motifs within a song and the maximum summation of the similarity values of the related motifs in each cluster.

The rest of this paper is organized as follows. In Sect. 2, we briefly describe the fundamental features of music and the general definition of theme melody. We also discuss the previous related works on the extraction of theme melodies from a song. In Sect. 3, we introduce a theme melody extraction mechanism that uses the modified graphical clustering algorithm. Also, we describe how the theme melodies that are approximately repeated within a song can be effectively extracted from a song for the content-based music information retrievals. Section 4 concludes this paper with the description of future works.

2 Related Works

2.1 The Basic Concept of Theme Melody

Music listeners get the semantics of music from the sequence of notes. The note that organizes music basically has four characteristics such as pitch, intensity, duration, and timbre. A music composed from the continuous notes has three characteristics: rhythm, melody, and harmony. Among the above characteristics of music, we adopt the melody as the primitive feature of music for supporting content-based music information retrieval because the melody can be a decisive component for representing music ([6,7,8]).

As we described above, music has a pattern of notes, i.e. melodies as the important component. A note alone can not be a pattern. As notes are added, the duration relationship between continuous notes is accomplished and finally a melody becomes clear. The pattern generally has a hierarchic structure that allows listeners to understand the music ([9]).

The motif is the minimum pattern that has some meaning by itself within a song. The theme is one of the important motifs that have the characteristic, 'theme reinstatement' that means the theme must be repeated more than once with some variances within a song ([8,9]). According to the composer and the song, the degree of variances of the theme melodies varies. In general, music listeners remember the theme melodies of a song as the main meaning of the song.

2.2 Previous Works on Theme Extraction from Music

To improve the performance of the content-based music retrieval systems, index mechanisms that contain the representative parts, theme melodies, of music as the indexing terms have been proposed. For that, several researches ([3,11]) have proposed the extraction mechanisms of theme melodies from music objects. Among them, in this section, we will discuss the mechanism proposed in [3] since this mechanism is considered as an advanced mechanism.

In this mechanism, a melody is represented as the string of absolute pitches of notes in the melody. For a sub-string Y of a music feature string X, if Y appears more than once in X, we call Y a repeating pattern of X. The frequency of the repeating pattern Y, denoted as *freq*(Y), is the number of appearances of Y in X.

Consider the melody string "do-re-mi-fa-do-re-mi-do-re-mi-fa", this melody string has ten repeating patterns. However, the repeating pattern "re-mi-fa", "mi-fa", and "fa" are sub-strings of the repeating pattern "do-re-mi-fa" and *freq*("do-re-mi-fa") =

freq("re-mi-fa") = *freq*("mi-fa") = *freq*("fa") = 2. Similarly, the repeating patterns "do-re", "re-mi", "do", "re", and "mi" are sub-strings of the repeating pattern "do-re-mi" and the *freq*("do-re-mi") = *freq*("do-re") = *freq*("re-mi") = *freq*("do") = *freq*("re") = *freq*("mi") = 3. Among the ten repeating patterns of the melody string, only "do-re-mi-fa" and "do-re-mi" are considered as non-trivial. The non-trivial patterns in a song are considered as the theme melodies of the song.

By using this mechanism, the exactly repeated longest patterns of a music object are considered and extracted as the theme melodies of the music. In other words, the previous mechanism does not consider the approximately (not exactly) repeating pattern in extraction of theme melodies. In general, however, a theme melody can be approximately repeated more than once with some variations of pitches and lengths of notes in the melodies. Therefore, for effective content-based music retrieval, we need a theme melody extraction mechanism that can deal with the some variation of theme melody during extracting theme melodies from music object.

3 An Extraction Mechanism of Theme Melodies from Music

3.1 The Procedure of Theme Melody Extraction Mechanism

The procedure of the theme melody extraction mechanism proposed in this paper is shown in Fig. 1. When a music file is submitted, it is decomposed into the set of motifs each of which is the minimum meaningful unit. From motifs, pitches and lengths of notes are extracted. Then the mechanism computes the similarity values between all pairs of motifs of the song by using the similarity computation equation proposed in [6]. At next, the similarity matrix (or the similarity graph) is constructed. By using the proposed graphical clustering algorithm that is described in Sect. 3.2, the motifs of the song are clustered based on the similarity values between two motifs, of the similarity matrix. Finally, a representative melody from each cluster is selected based on the locations of the melodies in the song and the maximum summation of the similarity values concerned to the melody in a cluster.

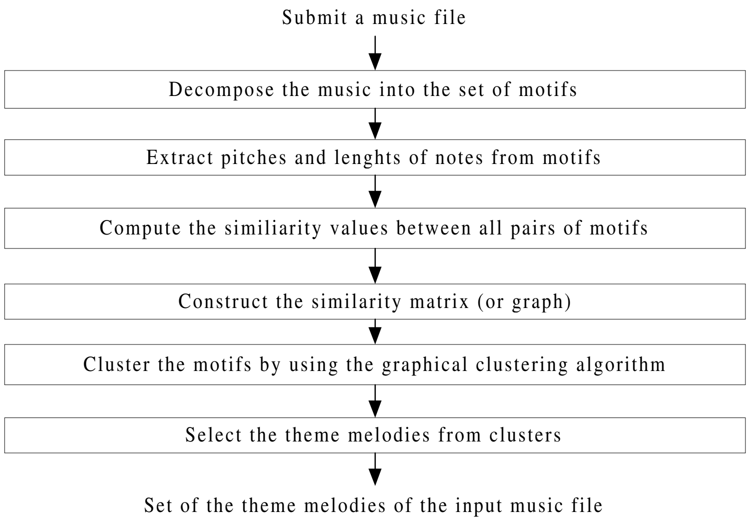

Submit a music file
Decompose the music into the set of motifs
Extract pitches and lenghts of notes from motifs
Compute the similiarity values between all pairs of motifs
Construct the similarity matrix (or graph)
Cluster the motifs by using the graphical clustering algorithm
Select the theme melodies from clusters
Set of the theme melodies of the input music file

Fig. 1. The Procedure of Theme Melody Extraction Mechanism

As an example, we consider a Korean children song of Fig. 2. The song is divided into 8 motifs. The motif is labeled with the circled number in Fig. 2.

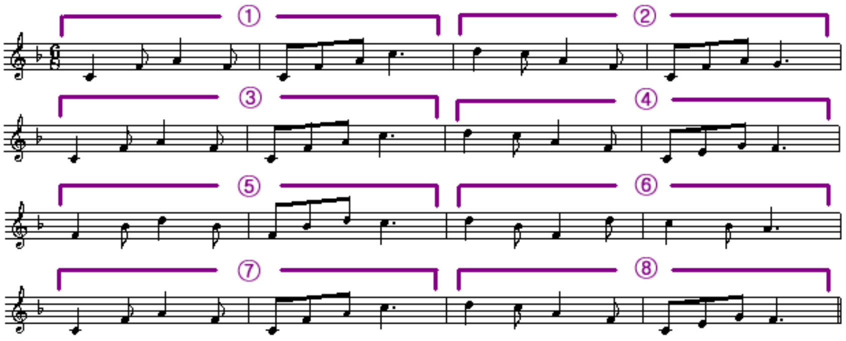


Fig. 2. The Score of a Korean Children Song “Nabi Nabi Hin-Nabi”

The similarity values between all pairs of motifs are computed and the similarity matrix is shown in Table 1. The entry (i, j) of i-th row and j-th column in the similarity matrix stands for the similarity value between i-th motif and j-th motif of the song. From the score of Fig. 2, since we can recognize that first and third and seventh motifs are exactly same to each other, the entries (1, 3), (1, 7), and (3, 7) have 100 as the similarity values. Also since the similarity matrix is of triangular matrix, the entries (3, 1), (7, 1), and (7, 3) have 100 too.

Table 1. The Similarity Matrix for "Nabi Nabi Hin-Nabi"

Motifs	1	2	3	4	5	6	7	8
1	100	85	100	82	86	62	100	82
2	85	100	85	96	81	73	85	96
3	100	85	100	82	86	62	100	82
4	82	96	82	100	77	70	82	100
5	86	81	86	77	100	68	86	77
6	62	73	62	70	68	100	62	70
7	100	85	100	82	86	62	100	82
8	82	96	82	100	77	70	82	100

3.2 A Graphical Clustering Algorithm

To form a fragment in database design, [12] proposed a graphical clustering algorithm that subdivides the attributes of a relation into groups of the attributes based on the attribute affinity matrix, which is generated from the attribute co-relationships.

Since theme melody extraction mechanism needs clustering algorithm that clusters motifs into groups of the motifs based on the similarity matrix doing like in the fragmentation in database design, with the slight modification, the graphical clustering algorithm proposed in [12] can be applied to the theme melody extraction problem.

To describe the modified graphical clustering algorithm in detail, we first introduce the notations and terminologies that are used in [12].

- Capital letters or numbers denote nodes, i.e., motifs of a song.
- Lowercase letters or parenthesized two node numbers such like (1-2) denote edges between two nodes.
- P(*e*) denotes the similarity value of edge *e* between two motifs.
- *Primitive cycle* denotes any cycle in the similarity graph.
- *Cluster cycle* denotes a primitive cycle that contains a cycle node. In this paper, we assume that a cycle stands for a cluster cycle unless otherwise stated.
- *Cycle completing edge* denotes an edge that would complete a cycle.
- *Cycle node* is that node of the cycle completing edge, which was selected earlier.

- *Former edge* denotes an edge that was selected between the last cut and the cycle node.
- *Cycle edge* is any of the edges forming a cycle.
- *Extension of a cycle* refers to a cycle being extended by pivoting at the cycle node.

Based on the above definition we discuss the mechanism of forming cycles from similarity graph. As an example with Fig. 3, suppose edges *a* and *b* were selected already and *c* is selected next. At this time, since selection of *c* forms a primitive cycle, (*a, b, c*), we have to check the possibility of a cycle, i.e., whether it is a cluster cycle. Possibility of a cycle results from the condition that no former edge exists, or P(former edge) ≤ P(all the cycle edges). The primitive cycle (*a, b, c*) is a cluster cycle because it has no former edge. Therefore, the cluster cycle (*a, b, c*) is marked as a candidate cluster and node *A* becomes a cycle node.

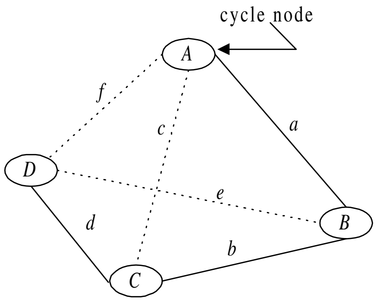


Fig. 3. Cycle Detection and Its Extension([12])

Let us explain how the extension of a cycle is performed. In Fig. 3, after determining *A* is a cycle node, suppose edge *d* is selected. Then, *d* should be checked the possibility of extension of the cycle, i.e., whether it is a potential edge for the extension of the cycle (*a, b, c*). Possibility of extension results from the condition of P(edge being considered or cycle completing edge) ≥ P(any one of the cycle edges). Thus, the old cycle (*a, b, c*) can be extended to the new one (*a, b, d, f*) if the edge *d* under consideration or the cycle completing edge *f* satisfies the possibility of extension: P(*d*) or P(*f*) ≥ minimum of (P(*a*), P(*b*), P(*c*)). Now the process is continued: suppose edge *e* is selected next. But we know from the definition of the *extension of a cycle* that *e* cannot be considered as a potential extension because the primitive cycle (*d, b, e*) does not include the cycle node *A*. Hence it is discarded and the process is continued.

We explain the relationship between a cluster cycle and a partition. There are two cases in partitioning. In both bases, after partitioning, the threshold is updated to this cutting value when this cutting value is greater than the saved threshold.

1. Creating a partition with a new edge. If a new edge (e.g. *d* in Fig. 3) by itself does not satisfy the possibility of extension, we continue to check an additional new edge called cycle completing edge (e.g. *f* in Fig. 3) for the possibility of extension. In Fig. 3, new edges *d* and *f* would potentially provide such a possibility of extension of the earlier cycle (*a, b, c*). If edges *d* and *f* meet the condition for the possibility of extension above (i.e., P(*d*) or P(*f*) ≥ minimum of (P(*a*), P(*b*), P(*c*)), then the extended new cycle contains edges *a, b, d, f*. If the condition is not satisfied, we produce a cut on edge *d* (called a cut edge) isolating the cycle (a, b, c) and this cycle is considered a candidate cluster.
2. Creating a partition with a former edge. After cutting in the previous case, if there is a former edge, then change the previous cycle node to that node where the cut edge is incident, and check for the possibility of extension of the cycle by the former edge. For example, in Fig. 4, suppose that *a, b*, and *c* form a cycle (*a, b, c*) with cycle node *A*, and that there is a cut on *d*, and that the former edge *w* exists. Then the cycle node is changed from *A* to *C* because the cut edge *d* originates from *C*. We are now evaluating the possibility of extension of the cycle (*a, b, c*) into one that contains the former edge *w*, i.e., the cycle (*a, b, e, w*). If *w* or *e* does not satisfy the possibility of extension, i.e., P(*w*) and P(*e*) < minimum of (P(*a*), P(*b*), P(*c*)), then the result is the following, w is declared as a cut edge, node C remains as the cycle node, and edges *a, b, c* becomes a partition. Otherwise, if the possibility of extension is satisfied, the result is the following, cycle (*a, b, c*) is extended to cycle (*e, w, a, b*), node *C* remains as the cycle node, and no partition can yet be formed.

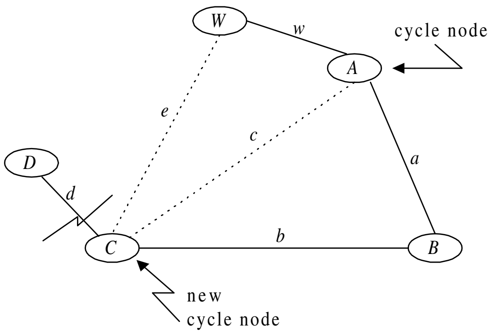


Fig. 4. Partition ([12])

In post-processing, the meaningless clustering edges are eliminated from the candidate clusters by using the threshold value that is continuously updated during partitioning describe above. Here, the meaningless clustering edge is an edge having smaller similarity value then the final threshold. After the post-processing, a cluster in which no meaningful edge exists removed from the set of clusters.

After post-processing, the theme extraction mechanism selects a motif from each cluster as the representative of the cluster based on the locations of motifs and the similarity values in the cluster. In general, human is likely to remember the first motif of a song as a theme melody of the song. Hence, in our mechanism, from the cluster in which the first motif is included, we select the first motif as the representative of the cluster. Otherwise, i.e., from the cluster in which the first motif is not included, we select a motif that has the maximum summation of the similarity values of all adjacent edges to the motif node as the representative of the cluster.

The modified graphical clustering algorithm that can be used for clustering motifs based on the similarity matrix is in Fig. 5.

3.3 Experimentation of the Theme Melody Extraction Mechanism

As an example for showing the effectiveness of the theme melody extraction mechanism proposed in this paper, let's consider the similarity matrix of Table 1 for a Korean children song of Fig. 2. Since the similarity matrix of Table 1 can be considered as the fully connected similarity graph as shown in Fig. 6a, in the following example we use the similarity graph of Fig. 6a instead of the similarity matrix of Table 1. Due to the space restriction, we do not present all similarity values in similarity graphs, however when we need the similarity value of an edge we present the value as the label on the edge.

From the node numbered 1, the graphical clustering algorithm starts. Among the edges that are adjacent to node 1, the edge (1-3) between node 1 and 3 is selected since it has the maximum similarity value. Since selecting edge (1-3) does not satisfy the condition of step 3, i.e., it does not forms a primitive cycle, and the condition of step 4, i.e., there is not a candidate partition, step 2 of Algorithm 1 selects edge (3-7) as the next edge for the next iteration. Since edge (3-7) does not satisfy the conditions of step 3 and step 4 again, edge (7-1) is selected as the next edge based on the conditions described at step 2. Since selecting edge (7-1) forms a primitive cycle (1-3, 3-7, 7-1) (see Fig. 6b), i.e., satisfying the condition of step 3 of Algorithm 1, step 3 is executed. Since there is no cycle node, the "possibility of a cycle" describe in Sect. 3.2 should be checked. Since there is no former edge, the condition of the possibility of cycle becomes true and the primitive cycle is considered as the cluster cycle and also becomes a candidate partition. So, node 1 becomes the cycle node of the primitive cycle (1-3, 3-7, 7-1). Then, edge (7-5) is selected as the next considering edge for the next iteration since it is adjacent to node 7 and, at the same time, it has the maximum similarity value among the edges adjacent to node 7. Since the selection of edge (7-5) satisfies the condition 4, i.e., edge (7-5) does not form a primitive cycle and a candidate cluster (1-3, 3-7, 7-1) exists and there is no former edge, the possibility of extension of the primitive cycle, i.e., whether edge (7-5) can enlarge the primitive cycle should be checked. Since edge (7-5) cannot enlarge the primitive cycle, the edge (7-5) is get cut and the primitive cycle becomes a candidate cluster and the threshold is set to 86 (notes that this cutting value is larger than the initial threshold 0). See Fig. 6d as the result of forming the first candidate cluster, (1-3, 3-7, 7-1) with the new threshold 86.

```
Algorithm 1: The Modified Graphical Clustering Algorithm
Input: Similarity Matrix of n ×n size or the fully connected similarity graph of n nodes
Output: Set of clusters that has similar motifs
Step 1. Start from the first node (row 1) with threshold = 0
Step 2. Select an edge that satisfies the following conditions and, when all nodes are
        used this iteration will end and go to step 5:
    (1)     If a cutting occurs in the previous iteration, starts from the next node adjacent
            to the cutting edge.
    (2)     Otherwise, it should have the largest value among the possible choices of
            edges be linearly connected to the tree already constructed
Step 3. When the next selected edge forms a primitive cycle:
        (1) If a cycle node does not exist, check for the "possibility of a cycle" and if the
            possibility exists, mark the cycle as a cluster cycle with the cycle node. Con-
            sider this cycle as a candidate partition. Go to step 2
        (2) If a cycle node exists already, discard this edge and go to step 2
Step 4. When the next selected edge will not form a cycle and a candidate partition ex-
        ists:
        (1) If no former edge exists, check for the possibility of extension of the cycle by
            this new edge. If there is no possibility, cut this edge and consider the cycle
            as a partition. Cutting value becomes the new threshold when this cutting
            value is greater than the saved threshold. Go to step 2
        (2) If a former edge exists, change the cycle node and check for the possibility of
            extension of the cycle by the former edge. If there is no possibility, cut the
            former edge and consider the cycle as a partition. Cutting value becomes the
            new threshold when this cutting value is greater than the saved threshold. Go
            to step 2
Step 5. Remove the meaningless edges that have smaller similarity values than the final
        threshold from the set of partitions.
```

Fig. 5. The Modified Graphical Clustering Algorithm

According to the step 2 of Algorithm 1, for the next iteration, edge (5-2) is selected in Fig. 6e since edge (5-2) has the maximum similarity value among the edges adjacent to the cut node 5. Then, edge (2-4) and (4-8) are selected in order since the selections of these edges do not satisfy the conditions of step 3 and step 4, respectively. However, when edge (8-2) is selected as the next edge by the step 2 as shown in Fig. 6f, the condition of step 3 is satisfied, i.e., edge (8-2) forms a primitive cycle (2-4, 4-8, 8-2). Since there is no cycle node yet, the possibility of cycle is checked. Since there is a former edge (5-2) of the primitive cycle and P(former edge) ≤ P(all the cycle edges), i.e., P(edge(5-2)) ≤ the minimum of P(edge(2-4), P(edge(4-8), and P(edge 8-2) is satisfied, the primitive cycle (2-4, 4-8, 8-2) becomes a cluster cycle and node 2 becomes the cycle node of the cluster cycle(see Fig. 6g). And, as the next edge, edge (8-5) is selected. Since edge (8-5) does not for a cycle and there is a candidate partition, i.e., the condition of step 4 is satisfied and there is a former edge (5-2), change the cycle node from node 2 to node 8. the possibility of extension of the cycle by the former edge should be checked, whether the former edge has the larger similarity value than the cycle edges. In Fig. 6h, since the similarity value 81 of the former edge (5-2) is not larger than the similarity values 96, 100, 96 of the cycle edges (2-4), (4-8), and (8-2), respectively, edge (5-2) becomes a cut edge and the cluster cycle (2-4, 4-8, 8-2) becomes a new candidate partition. However, the similarity value 81 of the cut edge is not greater than the saved threshold, the threshold is not updated, i.e., the threshold remains 86. Fig. 6i shows the second candidate partition formed.

The remaining nodes 5 and 6 form a third candidate partition as shown in Fig. 6j. Hence, from the similarity graph of a Korean children song in Fig. 2, we can form 3 candidate partitions finally as shown in Fig. 6k and the final threshold is 86.

In post-processing describe in step 5 of Algorithm 1, the edge (5-6) is removed from the third candidate partition, since it has smaller similarity value than the final threshold. The third candidate partition (5, 6) is eliminated. As the final result, two candidate partitions (1, 3, 7) and (2, 4 8) are returned from Algorithm 1.

From the final result of Algorithm 1, node 1 and 8 are selected as the representative melodies of the first candidate cluster (1, 3, 7) and the second candidate cluster (2, 4, 8), respectively, since node 1 is the first motif of the song and the first motif give human more impression than others generally and the node 8 has greater summation of the similarities of adjacent edges to that than others. Hence, the first motif and the last motif are selected as the theme melodies of a Korean children song. From the score of the song in Fig. 2, we can recognize the first and last motifs have exactly and approximately repeated several times to give users the impressions as the main melodies of the song.

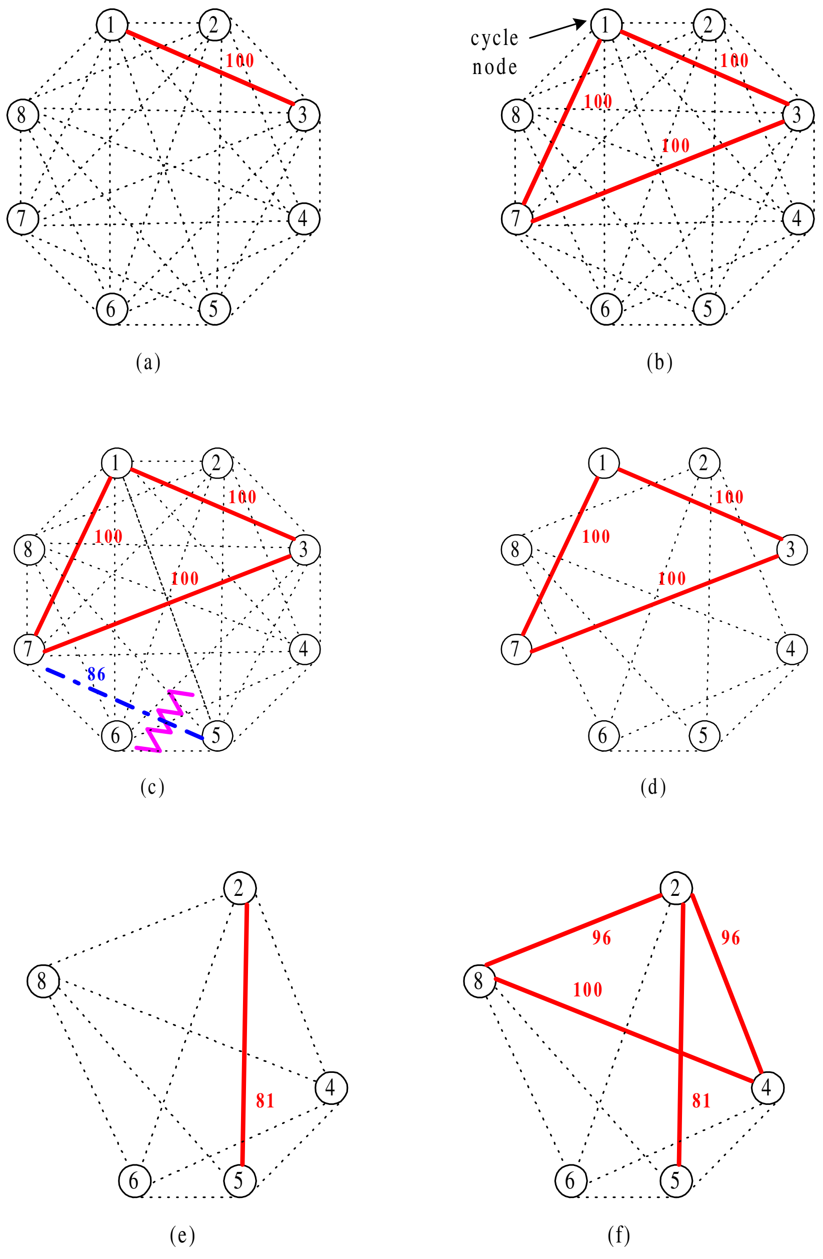


Fig. 6. Forming Candidate Clusters from a Similarity Graph

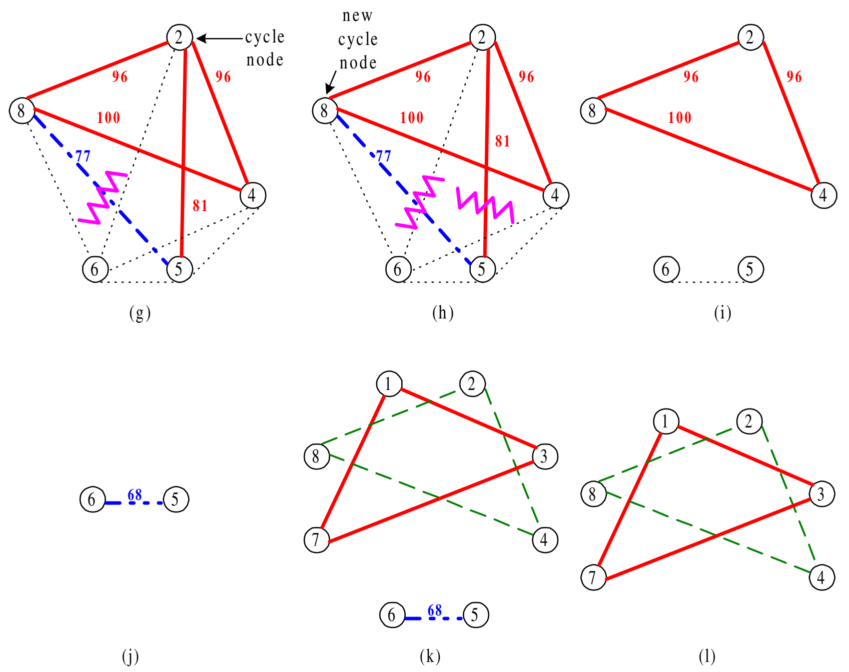


Fig. 6. Forming Candidate Clusters from a Similarity Graph (continued)

4 Conclusions

In this paper, we proposed a theme melody extraction mechanism in which a modified graphical clustering algorithm is used to cluster the motifs of a song into the set of similar melodies. In the first step of the extraction mechanism, we split a music file into the set of motifs each of which is the minimum meaningful unit. Then the system extracts pitch and duration information as the primitive features from the motifs. Then the system clusters the motifs into groups based on the similarity values calculated between all pairs of motifs based on the pitches and durations of notes of motifs such that each cluster has higher similarity values between them than others. For grouping motifs into set of clusters each of which includes similar motifs, we modified the graphical clustering algorithm proposed by [12] for database design. From the clusters, the system selects a theme melody based on the locations of motifs within a song and the maximum summation of the similarity value of the motifs in each cluster. As the experimental results, we showed an example in which we describe how the theme melodies of a song can be extracted by using the proposed algorithm.

Acknowledgements. This work was supported by grant No. 2000-1-51200-009-2 from the Basic Research Program of the Korea Science & Engineering Foundation.

References

1. A. Ghias, J. Logan, D. Chamberlin and B.C. Smith, "Query By Humming Musical Information Retrieval in an Audio Database," *ACM Multimedia*, 1995.
2. R. J. McNab, L.A. Smith, I. H. Witten, C. L. Henderson and S.J. Cunningham, "Towards the Digital Music Library: Tune Retrieval from Acoustic Input," *Digital Libraries*, 1996.
3. Chih-Chin Liu, Jia-Lien Hsu, and Arbee L. P. Chen, "Efficient Theme and Non-trivial Repeating Pattern Discovering in Music Databases", The Proceedings of the 15th International Conference on Data Engineering, 1999.
4. Ta-Chun Chou, Arbee L. P. Chen, and Chih-Chin Lie, "Music Databases: Indexing Techniques and Implementation", The Proceedings of IEEE International Workshop on Multimedia Database Manangement Systems, 1996.
5. Jia-Lien Hsu, Chin-Chin Liu, and Arbee L. P. Chen, "Efficient Repeating Pattern Finding in Music Databases", The Proc
6. Jong-Sik Mo, Chang-Ho Han, and Yoo-Sung Kim, "A Similarity Computation Algorithm for MIDI Musical Information", The Proceedings of 1999 IEEE Knowledge and Data Engineering Exchange Workshop", Chicago, USA, 1999.
7. So-Young Kim and Yoo-Sung Kim, "An Indexing and Retrieval Mechanism Using Representative Melodies for Music Databases", The Proceedings of 2000 International Conference on Information Society in the 21st Century, 2000.
8. Byeong-Wook Lee and Gi-Poong Park, "Everybody Can Compose Songs", Jackeunwoori Pub. Co., 1989.
9. Leonard B. Meyer, *Explaining Music: Essay and Explorations*, Saeguang Pub. Co., 1990
10. S. Wu and U. Mnaber, "Fast Text Searching Allowing Errors", Communication of ACM, Vol. 35, No. 10, 1992.
11. Arbee L. P. Chen, Jia-Lien Hsu, and C. C. Liu, "Efficient Repeating Pattern Finding in Music Databases", The Proceedings of ACM Conference on Information and Knowledge Management, 1998.
12. Shamkant B. Navathe and Minyoung Ra, "Vertical Partitioning for Database Desing: A Graphical Algorithm", The Proceedings of ACM SIGMOD, 1989.

A Multilingual Information System Based on Knowledge Representation

Catherine Roussey, Sylvie Calabretto, and Jean-Marie Pinon

LISI, Bât 501, INSA de Lyon, 20 avenue Albert Einstein F 69621 VILLEURBANNE Cedex
{catherine.roussey,sylvie.calabretto,jean-marie.pinon}@lisi.insa-lyon.fr

Abstract. This paper presents a new model for multilingual document indexing and information retrieval used in a documentary information system. Multilingual system exploits multilingual document collection which documents are written in different languages, though each individual document may contain text in only one language. We have developed a multilingual information retrieval system based on knowledge representation model. In order to carry on the implementation of our information retrieval system we chose an expressive formalism containing relation properties: the Sowa Conceptual Graph (CG). In this article, we define a new model of conceptual graph in order to enhance the effectiveness of our matching retrieval function, and we present the architecture of our multilingual information system which manages XML documents. Our approach has been applied to a collection of mechanical documents in a scientific library.

1 Introduction

This paper deals with a new approach for document indexing and retrieving in a multilingual information system. A documentary information system is a system managing and exploiting a document collection. By managing, we mean mainly all the operations linked to the storage and retrieval of the documents by an user. More precisely, a documentary information system performs different functions: the document storage, the document indexing and retrieval, the visualization of documents (visualization of entire document, abstract of document, etc.), the document edition, the navigation in the document collection and finally the management of versions and variants. A "multilingual information system" is a documentary information system exploiting multilingual document collection which documents are written in different languages, though each individual document may contain text in only one language. The paper focus on the indexing and information retrieval function in a multilingual document collection. We present a multilingual information retrieval system based on knowledge representation model. Following recent works, the semantic of index must be enhanced, so the usual list of keywords is transformed to a more complex indexing structure, where relations link keywords. Added semantic to index implies to propose an adapted matching function. The logical Information Retrieval (IR) model seems to be the only model suitable for managing complex semantic structures. Based on this model, several systems have been developed using different formalisms and different interpretations. In order to carry on the implementation of our logical IR system we chose an expressive formalism containing relation properties: the Sowa Conceptual Graph (CG) [6]. Different systems have used this formalism to implement a logical IR system. Moreover, one notes that these systems produce lot of silence that decrease the recall rate. In this article, we define a new model of conceptual graph in order to enhance the effectiveness of our matching retrieval function.

First of all, we present the Conceptual Graph formalism. Follows a recent state of the art about logical IR system based on CG. Afterwards, we introduce our model and the corresponding algorithms. And finally, we present our prototype and its evaluation.

2 Conceptual Graph Formalism

In 1986, van Rijsbergen [4] models the relevance of a document for query by a logical uncertainty principle. That is to say given two logical formulas: *d,* which is the document representation and *q* which is the query representation, a matching function between *d* and *q* measures the uncertainty of $d \rightarrow q$ related to a given data set *Ks*. Indeed, this function determines the minimal transformation necessary on a data set *Ks* to establish the truth of the implication $d \rightarrow q$. This approach improves the use of knowledge in IR System because document descriptors can be more than terms. So this model proposes a optimized matching function between document and query, taking into account the semantic associated to document descriptors.
Different interpretations about this logical principle are possible due to the lack of precision about the data set *Ks* [2]:

1. *Ks* can be general domain knowledge related to *d* and *q* . So we should evaluate the *Ks* minimal transformation to obtain the implication $d \rightarrow q$.
2. Ks can be fixed, *d* can be modified. So we should evaluate the minimal transformation *d* into *d'* in order to obtain the implication $d' \rightarrow q$.
3. *q* can be changed. So we should evaluate the minimal transformation *q* into *q'* in order to obtain the implication $d \rightarrow q'$.

In fact, Rijsbergen propose general model where any logic can be applied. Thus, any formalism can be chosen, since it has a logical implication. Some recent works [1, 3] use the formalism of Sowa Conceptual Graph to implement a logical IR system.

A conceptual graph is an oriented graph composed of concept nodes, conceptual relation nodes (or relation node) and edges between concept and relation nodes. A concept is labeled by a *type* and possibly a *marker*. *Type* corresponds to a semantic class and *marker* is a particular instance of a semantic class. For instance, [MAN: *] stands for the concept of all possible men. This concept is called a generic concept also noted [MAN]. On the other hand, [MAN: Bill Clinton] stands for the concept of a man named Bill Clinton. "*" and "Bill Clinton" are two examples of *marker*. A relation node is only labeled by a type. A specialization relation , noted $\leq$ classifies concept types and relation types, which link a generic type to a more specific one. A relation type has a fixed number of arguments called *arity*. *Arity* is the number of concepts linked by this relation type. Each relation type has a *signature* that defines the most generic concept type usable as an argument of the relation.

Specialization relations are useful to compare graphs by the Sowa projection operator. This operator defines a specialization relation between graphs. As shown in Fig. 1, there is a projection of a graph *H* in a graph *G* if there exists in *G* a copy of the graph *H* where all nodes are specialization of *H* nodes.

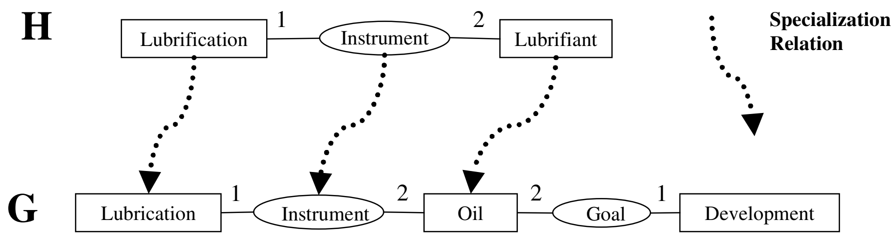


Fig. 1. A projection example

Moreover, Sowa proposes a translation mechanism (the *Φ* operator) from CG into first order logical formula. For each graph *g*, is associated a logical formula *Φ(g)*. Sowa has proven that there is a relation between the existence of a projection between two graphs and the implication of their logical formulas. There is a projection of a graph *q* in a graph *d* if the formula *Φ(d)* associated to *d* implies the formula *Φ(q)* associated to *q: Φ(d) → Φ(q)*. Therefore, conceptual graph formalism is used to implement logical operational systems where the matching function is based one the projection operator. For example, we could consider the previous graph *H* as a representation of a query and the graph *G* corresponds to the index of document. There is a projection of *H* in *G* is equivalent to *Φ(G) → Φ(H)* so the document is relevant for the query.

3 Related Works

Different logical Information Retrieval System, based on Conceptual Graph formalism, has been developed, depending on the interpretation of the uncertainty logical principle.

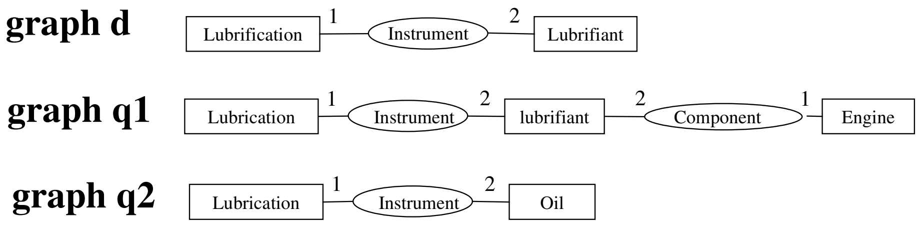


Fig. 2. Graphs examples for queries and documents

1. Ounis and all [3] have experienced the first interpretation of Rijsbergen principle in the RELIEF system. The set of domain knowledge Ks is transformed by using re-

lational properties such as symmetry, transitivity, inversion etc…This relation properties are all captured in Ks in order to refine the indexes and thus improve retrieval effectiveness. Another contribution of this work is to propose a fast matching function during execution time, even if in graph theory a projection can not be performed in polynomial time. Thanks to inverted file and acceleration tables, RELIEF computes some pre-treatments during indexing time, so projections are performed faster during retrieval.
2. David Genest [1] has noted that one of the drawbacks of the projection operator is to increase the silence. One of the reasons is that matching function based on projection give boolean results. There is a projection from a query to a document or there is not. Moreover documents which are judged relevant are only specialization of the query. For example, considering Fig. 2 there doesn't exist a projection from *q1* or *q2* into *doc*. However if *q1* and *q2* represent queries and *doc* represents a document index, this document seems to be relevant for these queries. In order to take in account such problems, David Genest proposes an implementation of the second interpretation of uncertainty principle. He defines some transformations on Conceptual Graph used as document index in order to find a projection from the query graph to the index graph. These transformations include specialization or generalization of node labels, node joints or node and edge additions. Moreover a mechanism is proposed to order sequence of transformations. As a consequence, the matching function based on projection become a ranking function and orders relevant documents for a query.
3. The third interpretation of the uncertainty principle implies to transform query to establish the implication $d \rightarrow q$. Since the projection operator implies expensive treatments, it doesn't seem operational to process it during retrieval, in real time.

Our proposition takes Ounis and Genest improvements into account, by proposing a graph matching function optimized for the information retrieval needs. Now, we present our graph formalism called semantic graph.

4 Semantic Graph Model

Ontology enables to define the graph vocabulary. To simplify the formalism, the notion of markers is eliminated from the conceptual graph model, because in our case, document index contain only generic notion.

An ontology O is a 3-tuple $O = (T_C, T_R, \sigma)$ where:

- T_C is a set of concept type partially ordered by the specialization relation noted $\leq$.
- T_R *is a* set of binary relation types[1] partially ordered by $\leq$
- σ, called signature, is a mapping which associates with any relation types the greatest concept type of its arguments. In other words, for any $t_r \in T_R$, the type of the k^{th} argument of t_r should be more specific than the type of the k^{th} argument of $\sigma(t_r)$. The i^{th} argument of $\sigma(t_r)$ is noted $\sigma_i(t_r)$.

[1] In general, a type of relation can have any arity, but in this paper, relations are considered to be only binary relations like case relations or thematic roles associated with verbs [7].

Afterwards, we present our model called semantic graph. Comparing to Conceptual Graph, the accent is made upon relation between concepts. All concepts should be linked to at least another concept to build an arch.

A semantic graph is a 4-tuple $Gs = (C, A, \mu, \nu)$ where :

- C is a set of concept nodes contained in Gs.
- $A \subset C \times C$ is a set of arches contained in Gs. For each arch $a= (c, c') \in A;$ the ith concept node of a (also called argument of a) is noted a_i: $(a_1 = c$ and $a_2 = c')$. The set of concept nodes (argument) belonging to one of the arches of A is noted A_i.
- $\mu: C \rightarrow T_C$, μ is an application, which associated for each concept node, $c \in C,$ a label $\mu(c) \in T_C$, $\mu(c)$ is also called the *type* of c.
- $\nu: R \rightarrow T_R$, ν is an application, which associated for each arch, $a \in A,$ a label $\nu(a) \in T_R$. $\mu(c)$ is also called the *type* of a.

A semantic graph checks some constraints :

1. For each $a \in A / a = (c, c'),\ \nu(a)=r\ ,\ \mu(a_i) \le \sigma_i(r)$ so $\mu(c) \le \sigma_1(r)$ and $\mu(c') \le \sigma_2(r)$.
2. All the nodes concepts belong to at least one arc, so $C \subset A_1 \cup A_2$

The pseudo projection operator is an extension of the projection operator of Sowa Conceptual Graph. A pseudo projection defines morphism between graphs with less constraint than the projection operator does. The existence of a pseudo projection of a graph H in a graph G is equivalent to the fact that a part of the information represented by G is close to the information represented by H.

4.1 Pseudo Projection Operator:

A pseudo projection from a semantic graph $H = (C_H, A_H, \mu_H, \nu_H)$ to a semantic graph $G = (C_G, A_G, \mu_G, \nu_G)$ is a pair of mapping $\Pi= (f,g)$, such as $f: A_H \rightarrow A_G$, associates an arch of H with an arch of G and $g: C_H \rightarrow C_G$ associates a concept node of H with a set of concept nodes of G. Π has the following properties:

1. Concept nodes can not be preserved.
   For any concept nodes $c, c' \in C_H$, it is possible that $g(c) = g(c')$.
   Two concept nodes of H can have the same image by Π (for example, a same node of G).
   For any concept nodes $c \in C_H$, it is possible that $g(c) = \{b, b'\}$ such that b and $b' \in C_G$.
   A concept node of H can have several images by Π (two distinct concept nodes of G).
2. Type of concept nodes can be restricted.
   For any concept node c of C_H, if there exists $b \in C_G$ such that $b \in g(c)$ then $\mu_G(b) \le \mu_H(c)$.
   For any concept node belonging to the graph H, the type of its image per Π is more specific than its type.
3. Type of concept node can be increased.
   For any concept node c of C_H, if there exists $b \in C_G$ such that $b \in g(c)$ then $\mu_H(c) \le \mu_G(b)$.

For any concept node belonging to the graph *H*, its type is more specific than the type of its image per *Π*.

4. Arches are preserved.
   For any arch $a = (c, c')$ of A_H there exists $f(a)=(b, b')$ an arch of A_G such that $b \in g(c)$ and $b' \in g(c')$.
5. Type of arch can be restricted.
   For any arch label $v_H(a), f(v_G(a)) \leq v_H(a)$
   For any arch belonging to the graph H, the type of its image per *Π* is more specific than its type.
6. Type of arch can be increased.
   For any arch label $v_H(a),\ v_H(a) \leq f(v_G(a))$
   For any arch belonging to the graph H, its type is more specific than the type of its image per *Π*.

Indeed, pseudo projection operator enables to see if all the information represented by *H* are close to some information represented by *G*. Now we should define a mechanism enabling to see if part of the information from *H* are close to some information from *G*. At this point, the document *doc* of the Fig. 2 answers the query *q1*, but *doc* still be not relevant to *q2*.

4.2 Partial Pseudo Projection Operator

There is a partial pseudo projection from *H* to *G* if there is *H'*, a subgraph of *H* such as there exists a pseudo projection from *H'* to *G*.

Let us point out that, the pseudo projection operator does not preserve the number of concept nodes, because we focus on relation between concepts, so arches are more important than concepts. Moreover we think that it is no need to preserve the whole structure of a graph for IR purpose because semantic graph is equivalent is considered to be equivalent to its set of arches where concepts are unique.

From specialization relation, similarity functions between types will be defined. Consequently, evaluation of the similarity between two arches will be possible. Then, an evaluation of the pseudo projection operator between graph will enable us to define the function of similarity between two semantic graphs.

4.3 Similarity Function between Types

The specialization relation enables to define a similarity function between types, noted *sim*. *sim* is an asymmetrical function and it returns a real value ranging from 0 to 1.

$sim: T_C \times T_C \cup T_R \times T_R \rightarrow [0..1]$

sim is defined as follow:

– If two types are identical then the similarity function returns value 1.
– If a type t_1 specializes another type t_2 directly, i.e. there is not intermediate type between t_1 and t_2 in the type hierarchy, then the similarity function returns a constant value, fixed arbitrarily, lower than 1 such as: $sim(t_2, t_1) = V_S$ and $sim(t_1, t_2) = V_G$

– If a type t_1 specializes another type t_2 directly, i.e. there is an intermediate type t between t_1 and t_2 in the type hierarchy then the similarity function between t_1 and t_2 is the product of the similarity functions between (t_1, t) and (t, t_2) that is to say : $sim(t_2, t_1) = sim(t_2, t) \times sim(t, t_1)$ and $sim(t_1, t_2) = sim(t_1, t) \times sim(t, t_2)$.

4.4 Similarity Function between Arches

The similarity function between types enables to define a similarity function between two arches, noted Sim_A. Sim_A is a float function and it returns a value ranging from 0 to 1. This function is defined as follows:

a_H is an arch such as $a_H = (c_H, c'_H)$ and $\nu(a_H) = r_H$ and a_G is an arch such as $a_G = (c_G, c'_G)$ and $\nu(a_G) = r_G$.

$$Sim_A\ \ (a_H\ ,\ \ a_G)\ !\ \ \frac{sim\ \ \#_H\ \ a_H\ \ ,\ \ \#_G\ \ a_G\ \ \forall\ \ \mathop{}\limits_{i!1}^{2}\ sim\ \ a_{Hi}\ ,\ \ a_{Gi}}{3} \tag{1}$$

or

$$Sim_A\ \ (a_H\ ,\ \ a_G)\ !\ \ \frac{sim\ \ r_H\ ,\ \ r_G\ \forall\ \ sim\ \ \exists\ c_H\ \ ,\ \ \exists\ c_G\ \ \ \forall\ sim\ \ \exists\ c'_H\ \ ,\ \ \exists\ c'_G}{3} \tag{2}$$

Indeed, Sim_A compute the average of the similarity between each arch component.

4.5 Pseudo Projection Evaluation

Considering two semantic graphs, pseudo projection from one graph to the other one is possible or not. Rather than a boolean result, a real value is preferred so a computation is proposed to evaluate the resemblance between two graphs.

If there is a pseudo projection Π from a graph $H = (C_H, A_H, \mu_H, \nu_H)$ to a graph $G = (C_G, A_G, \mu_G, \nu_G)$, then this pseudo projection is evaluated by a real function, noted val, defined as follows.

$$val\ \ \&\ \ \ !\ \ \frac{\underset{a\%A_H}{}\ \ sim_A\ \ a\ \ ,\ \ \ f\ \ a}{|A_H|} \tag{3}$$

val is the average of the similarity function between each arch of H and its image in G by Π.

4.6 Similarity Function between Graphs

The similarity function, noted sim_G, between a graph $H = (C_H, R_H, U_H, \mu_H, \nu_H)$ and a graph $G = (C_G, R_G, U_G, \mu_G, \nu_G)$ is a real function returning values ranging from 0 to 1.

$$Sim_G\,(H, G) = max(val(\Pi)). \tag{4}$$

Or

$$sim_G\ \ H, G\ \ \ !\ \ \frac{MAX_{\&}\ \ \ \ \underset{a\,\%A_H}{}\ \ sim_A\ \ a\ \ ,\ \ \ \&\ a}{|A_H|} \tag{5}$$

where Π *is a partial pseudo projection from H to G.*

5 Indexing and Information Retrieval Algorithms

After introducing our model of semantic graph, we shall concentrate on the organization of the indexes for faster retrieval. Our structure consists on an inverted file and several acceleration tables (see Fig. 3). The inverted file groups in the same entry all the documents indexed by an arch. This means that, given an arch, we can immediately locate the document indexed by that arch. This is the basis of the inverted file construction.

The acceleration tables enables to store the component of the arches and the semantic values between components. Indeed, The acceleration tables pre-compute all the similarity values of possible query arches before interrogation. The construction of the inverted file and the acceleration table is done off-line, as part of the indexing procedure.

5.2 Indexing Algorithm

```
Doc is a document. GraphIndex is the semantic graph in-
dexing Doc. O is an ontology. FirstArgValue is the ac-
celeration table containing the first argument (node
concept) of arches. SecondArgValue is the acceleration
table containing the second argument (node concept) of
arches. RoleValue is the acceleration table containing
the label of arches (relation). InvertedFile is the in-
verted file including - in the same entry - all the
documents indexed by an arch.

For each arch ArcDoc of GraphIndex do

         If NotFind(ArcDoc) then

             TArg1 is the type of the first concept argu-
ment of ArcDoc, TArg2 is the type of the second concept
argument of ArcDoc, TRelation is the type of the label
of ArcDoc
```

```
          For all TypeC related to TArg1 by the specialization relation in O do
              WeightArg1 ← SimC(TypeC, TArg1)
              FirstArgValue. AddTuple(ArcDoc, TypeC, WeightArg1)
          EndFor
          For all TypeC related to Targ2 by the specialization relation in O do
              WeightArg2 ← SimC(TypeC, TArg2)
              SecondArgValue. AddTuple (ArcDoc, TypeC, WeightArg2)
          EndFor
          For all TypeR related to TRelation by the specialization relation in O do
              WeightRel ← SimR(TypeR, TRelation)
              RoleValue.AddTuple(ArcDoc, TypeR, WeightRel)
          EndFor
      EndIf
      InvertedFile. AddTuple (ArcDoc, Doc)

NotFind(ArcDoc) is a boolean function, which returns
true if ArcDoc already exists in the Data Base.
```

An important part of the complexity of pseudo projection operator is done at indexing time. Because for each indexing component, we find all components which can be related to it (component can be specialized by or can specialize the indexing component) and then we compute the similarity between them. This makes it easier to compute the similarity between a query arch and an indexing arch.

5.2 Retrieval Algorithm

By making use of the pre-computations that are comprised in the acceleration tables, the retrieval function performs -in one operation- the evaluation of the pseudo projection between query graph and indexing graph.

```
GraphReq is a query graph and nbArc is the number of
arches in GraphReq. Threshold is a fixed threshold to
filter result documents. ListDocResult is a weighted
list of documents.

For each arch ArcReq of GraphReq do

          ListArcIndex ←FindRelatedArc(ArcReq)

          For each (ArcIndex, WeightArc) of ListArcIndex
do

           ListDoc ← FindDocList(ArcIndex)

// For a query arch, the document weight is the maximum
of similarity values between the query arch and one of
its indexing arch

           For each Doc of ListDoc do

             If ListDocArc.Belong(Doc) Then

               Weight ←ListDocArc.FindWeight(Doc)

               NewWeight ← max(Weight, WeightArc)

               ListDocArc.ReplaceWeight(Doc, NewWeight)

             Else

               ListDocArc.Add(Doc, WeightArc)

             EndIf

           EndFor

          EndFor

//For a query graph, the document weight is the sum of
all the similarities values (WeightArc) between a query
arch and one of its indexing arch divided by the number
of query arches.
```

```
          For each (Doc, WeightArc) of ListDocArc do
            If ListDocResult.Belong(Doc) Then
              Weight ← ListDocResult.FindWeight(Doc)
              NewWeight ← Weight + (WeightArc / nbArc)
              ListDocResult.ReplaceWeight( Doc, New-
Weight)
            Else
              ListDocResult.Add(Doc, WeightArc)
          EndFor
EndFor
For each (Doc, WeightArc) of ListDocResult do
          If WeightArc < Threshold Then
            ListDocResult.Remove(Doc, WeightArc)
EndFor
FindRelatedArc(ArcReq) is a function, which returns a
list (ListArcIndex), of arches (ArcIndex) associated
with a similarity values (WeightArc).
```

In the retrieval algorithm, the partial pseudo projections are obtained in polynomial time, as the most part of the algorithm consist in tables joint. Usually the cost of a projection operator between graph is over estimated, as it is often link to graph theory and want to find a morphism between indefinite structure. To over come this problem, we have limited the graph structure. We consider that a graph is a set of arches and that a concept node is unique in a semantic graph. Now we should evaluate if our limitation can obtain could result during a retrieval process.

6 The SyDoM Prototype

Our multilingual retrieval system is called SyDoM (Multilingual Documentary System)[5]. The system is implemented in JAVA on top of the Microsoft Access Database system. SyDoM is composed of three modules:

- The ontology module manages the documentary language with new vocabulary and new domain entity. Documentary language is used for indexing and querying a multilingual document collection.

- The indexing module indexes and annotates XML documents with semantic graphs using a set of metadata associated to the ontology.
- The retrieval module performs multilingual retrieval.

Actually the ontology can be displayed in different languages (French and English). This system performs also manual indexing process for XML documents using a set of metadata associated to the ontology. Our experimentation was done on English articles dealing with mechanics called pre-print of the Society of Automotive Engineers (SAE). The first step of the experiment is to build a domain ontology. Thanks to a mechanical thesaurus, we managed to have 105 mechanical concepts stored in our ontology. On top of that, we added 35 relations found in Sowa Knowledge Book [7]. During manual indexing only titles are taken in account. For our first experiments, we have manually indexed approximately 50 articles and used 10 queries. The average indexing graph consists of 4 arches and the average query graph consists of 2 arches.

The users introduce their queries through the query interface presented in Fig. 3. This figure correspond to the French query "modèle de combustion" (combustion model)

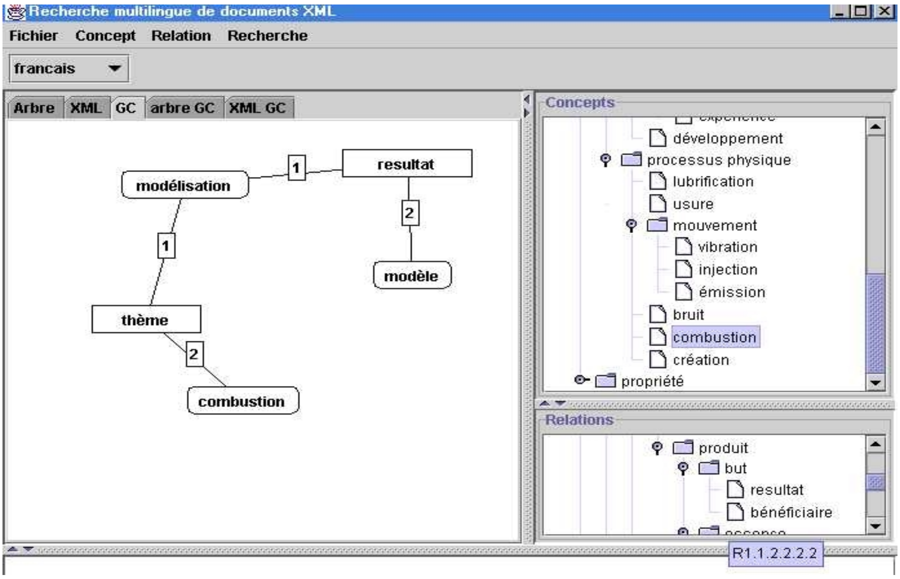


Fig. 3. SyDoM interface

When retrieval is started, the data introduced by the user are collected and an internal representation of the query is produced. The results are ordered from the most relevant document to the less pertinent one.

To evaluate our system, we compare it to the IR system used at the scientific library of our Institute, called Doc'INSA. In this system, documents and queries are represented by a list of keywords. The matching function between documents and queries evaluates the number of common keywords. The indices of this system were generated automatically from the index graphs of SyDoM, to avoid to take the index variability into account.

The next figure presents the performance of our system using different threshold (0.8, 0.6, 0.5, 0.4, and 0). We compute the average precision for ten recall intervals. The constant value of type similarity function are arbitrarily fixed ($V_G = 0.7$ *and* $V_S = 0.9$).

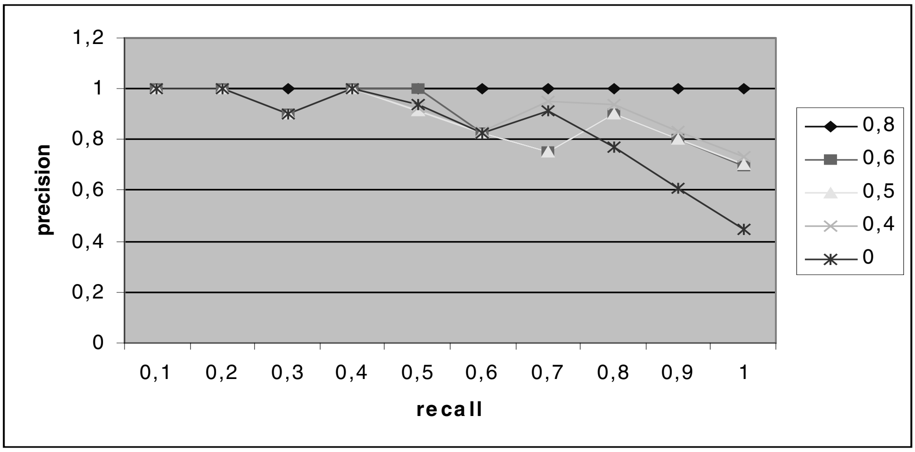


Fig. 4.: SyDoM evaluation using different thresholds

The trend of the curve can be explained by the fact that our collection size is small. Therefore, most part of the queries deals only with few documents. Because these documents are retrieved with an important weight (more than 0.8), the precision is good whatever the recall could be.

The next figure presents the comparison of SyDoM with the Doc'INSA system. We can noticed that relations treatment and hierarchy inference improve significantly the quality of the answer even for manual indexing.

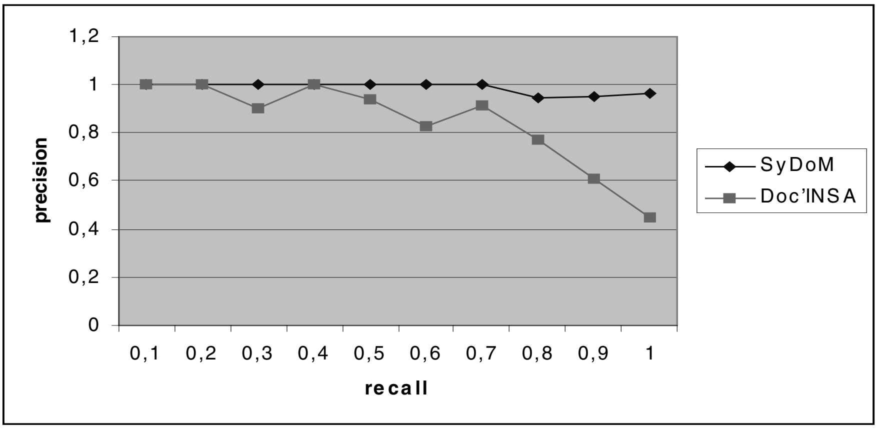


Fig. 5. Evaluation of SyDoM (threshold = 0.6) and Doc'INSA system

The next step would be to compare our system to RELIEF and David Genest One. We aim to experiment if our model -comparing to the extension of CG proposed by David Genest- could have similar results with less computational time. The main challenge is that David and I deal with manual indexing process so it is not easy to find human resources in order to carry on a real experiment.

7 Conclusion

In this paper, we proposed a prototype of a Multilingual Information System focusing on the indexing and information retrieval modules. We have defined a new model of conceptual graph in order to enhance the effectiveness of our retrieval system. We have, indeed, integrated a new extension of CG proposed by Genest and a fast retrieval technique. We have already noticed that our proposal is operational and give better results than traditional documentary system. This is very encouraging for a first implementation. At this point, we need to carry on further experiments.

References

1. D. Genest. « Extension du modèle des graphes conceptuels pour la recherche d'information ». PhD Thesis, Montpellier University, Montpellier, France 2000.
2. J.Y. Nie. « un modèle logique général pour les systemes de recherche d'informations. Application au prototype RIME ». PhD Thesis, Joseph Fouriei University, Grenoble, France 1990.
3. I. Ounis, M. Pasça. « RELIEF: Combining expressiveness and rapidity into a single system ». Proceeding of 18th SIGIR Conference, Melbourne, Australia, p 266-274, august 1998.
4. C.J. van Rijsbergen. « A new Theoritical Framework for Information Retrieval ». Proceeding of the 9th SIGIR Conference, Pisa, p 194-200m septembre 1986.
5. C. Roussey, S. Calabretto, J. M. Pinon « Un modèle d'indexation pour une collection multilingue de documents». Proceeding of the 3rd CIDE Conference, Lyon, France, p 153-169, July 2000
6. J. Sowa. « Conceptual Structures: information processing in mind and machine ». The System Programming Series, Addison Wesley publishing Company, 1984.
7. J. Sowa. « Knowledge Representation: Logical, Philosophical, and Computational Foundations ». Brooks Cole Publishing Co., Pacific Grove, CA., 2000.

Capturing Fuzziness and Uncertainty of Spatiotemporal Objects

Dieter Pfoser and Nectaria Tryfona

Computer Science Department, Aalborg University
Fredrik Bajersvej 7E, DK-9220 Aalborg East, Denmark
{pfoser,tryfona}@cs.auc.dk

Abstract. For the majority of spatiotemporal applications, we assume that the modeled world is precise and bound. This simplification seems unnecessary crude for many environments handling spatial and temporal extents, such as navigational applications. In this work, we explore fuzziness and uncertainty, which we subsume under the term *indeterminacy*, in the *spatiotemporal* context. We first show how the fundamental modeling concepts of *spatial objects*, *attributes*, *relationships*, *time points*, *time periods*, and *events* are influenced by *indeterminacy*, and then show how these concepts can be combined. Next, we focus on the *change* of spatial objects according to their geometry over time. We outline four scenarios, which identify discrete and continuous change, and we present how to model indeterminate change. We demonstrate the applicability of this proposal by describing the uncertainty related to the movement of point objects, such as the recording of the whereabouts of taxis.

1 Introduction

Spatiotemporal applications received a lot of attention over the past years. Requirements analysis [15], models [4], data types [8], and data structures [14] are some of the main topics in this area. Although considerable research effort and valuable results *do* exist, all the studies and approaches are based on the assumption that, in the spatiotemporal mini-world, objects have *crisp* boundaries, relationships among them are *precisely* defined, and *accurate* measurements of positions lead to error-free representations.

However, reality is different. Very often boundaries do not strictly separate objects but, rather, show a transition between them. Consider the example from an environmental system in which the different soil zones, such as desert and prairie, are not precisely bound. We encounter a transition, or *fuzziness*, between them. On the other hand, in navigational systems, the position of a moving vehicle, although precise in its nature, might not be exactly known, e.g., car A is in New York. We encounter *uncertainty,* i.e., *lack of knowledge* or *error* about its actual location.

In this paper, we deal with *fuzziness and uncertainty as related to spatiotemporal objects*. More specifically, we start by pointing out the semantic differences between the two cases that constitute *spatiotemporal indeterminacy*: *fuzziness,* concerning "blurry" situations, and *uncertainty,* expressing the "not-exactly-known" reality. We clarify these terms in the spatial and temporal domains, as well as the combined effect, i.e., spatiotemporal fuzziness and uncertainty. We show how the basic spatiotemporal modeling concepts of spatial objects, attributes, relationships, time points, time periods, events, and change are influenced by indeterminacy. We provide formal ways to describe this, while an example demonstrates the applicability of this proposal. A more elaborate discussion with the use of fuzzy set and probability theory in this area can be found in [16].

There are only few works towards spatiotemporal indeterminacy. [18] focuses on simple, abstract, spatial and temporal uncertainty concepts and integrates them to describe spatial updates in a GIS database. [13] discusses spatiotemporal indeterminacy for moving objects data. It is, however, limited to point objects and it does not take temporal errors into account. [2] aims at describing the change of fuzzy features over time using a raster representation. More work exists towards temporal, e.g., [5] and spatial indeterminacy, e.g., [1], [3], [7], [17], [19], [20].

The rest of the paper is organized as follows. Section 2 briefly presents the fundamental spatial and temporal concepts involved in the spatiotemporal application domain. Section 3 explores the semantics and gives the mathematical expression of indeterminate temporal concepts. Section 4 deals with indeterminate spatial concepts. Section 5 discusses change as the spatiotemporal concept affected by indeterminacy. Finally, Sect. 6 concludes with the future research plans.

2 Spatial and Temporal Concepts

To understand spatiotemporal indeterminacy, it is important to realize the fundamental spatial, temporal, and spatiotemporal concepts.

Spatiotemporal applications can be categorized based on the type of data they manage: (a) applications dealing with *moving* objects, such as navigational, e.g., a moving "car" on a road network, (b) applications involving objects located in space, whose characteristics and their position, may *change* in time, e.g., in a cadastral information system, "landparcels" change positions by changing shape, but they do not "move," and (c) applications integrating the above two behaviors, e.g, in environmental applications, "pollution" is measured as a *moving* phenomenon which *changes* properties and shape over time. The following modeling concepts are involved in environments like the aforementioned.

- *Spatial Objects* and their *geometry.* Spatial objects are objects whose position in space *matters*, e.g., a moving "car." Many times, *not only the actual* object's *position* matters, but its *geometry* does as well. For example, in a cadastral system the *exact geometry* of a "landparcel" is of importance. The geometry of the position of a spatial object can be (of type) point, line, region or any combination thereof [10].
- *Spatial Relationships.* Spatial relationships relate spatial objects, or more precisely, the positions of the objects, e.g., two landparcels share common borders.
- *Spatial Attributes* and their *geometry.* Spatial objects have, apart from descriptive attributes, also spatial attributes, e.g., the "vegetation" of a "landparcel." Values of spatial attributes depend on the referenced position and not on the object itself.

Spatial attributes are also related to geometries in space, as they split space in parts in whose extents the values of the spatial attributes remain the same; each part of space has (like, the objects' positions) geometry (of type) point, line, region or any combination thereof. For example, the attribute "vegetation" creates partitions of space with constant vegetation values in each partition, such as "forest", and "bushes." There are two types of spatial attributes, (a) those representing functions with continuous range, e.g., "temperature." Here the geometry of the partitions is point. (b) Those representing functions with discrete range, e.g., "vegetation" is represented as a set of regions. Not all spatial objects have spatial attributes. For example, no spatial attribute is usually assigned to a moving "car," while various ones, e.g., "soil type," may be assigned to a "landparcel."

- *Time.* Many different models of time exist. Some authors even propose taxonomies of time. In our work we assume a linear ordered time line, isomorphic to a finite subset of the natural numbers. The elements of this set are termed *chronons*.
- *Time points vs. Time periods.* Two basic models of time are used to record facts and information of a database: time point and time period. A time point t_1 is located during a chronon, while a time period $[t_k, t_m]$, with t_k, t_m time points and $k \leq m$ has a duration and is defined as a set of chronons.
- *Events* and *States*. An event occurs at an exact time point, i.e., an event has no duration. For example a "car crash.'' A state is defined for each chronon in a time point. It has duration, e.g., a "meeting" takes place from 9 am until 11am.

3 Temporal Indeterminacy

In temporal applications we are interested in events and their occurrence time. However, sometimes we only know approximately when an event occurred, e.g., a traffic accident happened between "2 pm and 4 pm." Next, we present models to represent indeterminacy in the temporal domain by adapting the model presented in [5].

3.1 Indeterminate Time Points

A time point is *determinate* if it is known during which chronon it is located. Fig 1a shows the determinate point I_1, based on the approach that a chronon is longer than a time point. A time point is *indeterminate* if it is not known exactly when, but approximately during which series of chronons it is located. An indeterminate time point is described by a *lower support*, an *upper support*, and a *probability function* [5]. The *supports* are chronons that delimit the location of the time point, e.g., for time point I_2 in Fig. 1a, the lower support is 5 and the upper support is 8; the probability function shows the likelihood where the time point is located within the range, e.g., in uniform distribution, it is equally likely for the time point to be located at chronons 5 to 8.

In the following, we use probability and fuzzy set theory to quantify indeterminacy. The probability mass function, p_x, for the indeterminate point x is

$$p_x(i) = P[x = i] : i \in \mathbb{N} \tag{1}$$

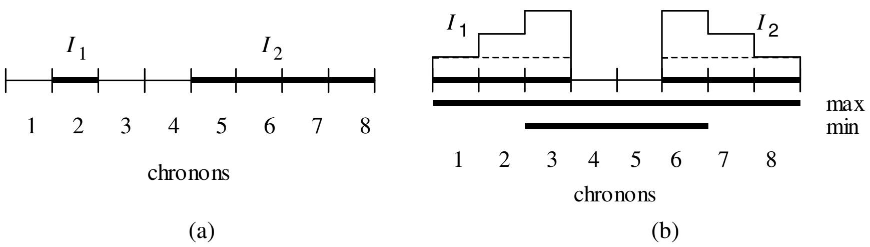


Fig. 1. (a) Determinate (I_1) and indeterminate (I_2) time points, (b) indeterminate time period, probabilities of bounding time points (solid line-probability density function, dashed line-probability mass function)

where $P[x=i]$ is the probability that the time point is located during chronon *i*. In our example, assuming uniform distribution, $P[I_2=6]=0.25$, the probability outside the range lower support–upper support is 0. Also, all indeterminate time points are considered to be independent, i.e.,

$$P[x=i \wedge y=j]=P[x=i]\times P[y=j] \tag{2}$$

We can state that all probability distributions are fuzzy sets [16]. By using the probability mass function as basis we obtain the following membership function

$$\mu_x(i)=\lambda p_x(i) \tag{3}$$

where λ is an arbitrary scale factor relating the membership grade to the probability.

3.2 Indeterminate Time Periods

A time period is a subset of the time line bound by two time points. Depending on whether the bounding points are determinate or indeterminate, we term the time period accordingly. In Fig. 2b, I_1 and I_2 denote the indeterminate start and end point of the period. Possible periods can range from chronon 1 to chronon 8 (max), but at least have to range from 3 to 6 (min).

The time period presented in Fig. 1b can also be perceived as having a fuzzy boundary. Next, we derive a membership function, $\mu_T(x)$, returning the degree to which an arbitrary chronon *x* is part of the time period *T*. From Fig. 1b, we can deduce that chronons 4 and 5 are definitely part of the time period *T*, whereas other chronons might be. Assuming a uniform distribution of the chronons within the time points I_1 and I_2, we can see that if chronon 2 is within the period so has to be chronon 3. Further, if chronon 1 is within, so have to be chronons 2 and 3. The same is true for chronons 6, 7, and 8 of I_2. Thus, in three cases chronon 3, in two cases chronon 2, and in one case chronon 1 is within period *T*. The probability mass function of I_1 and I_2 as shown in Fig. 1b gives the probability for a chronon to be in *T*. In summing up the probability from "the outside to the inside," we obtain a step function, the probability density function.

To derive the membership function, $\mu_T(x)$, we have to split the time period *T* into three parts; (1) the "core" (chronons 4 and 5), (2) the intervals I_1 and I_2, and (3) the outside world. A membership grade of 1 and 0 indicate definite and no membership in the time period, respectively. All chronons in the core have a grade of 1. The grade of the chronons in the intervals is equal to the value of the probability density function. Formula 4 summarizes the membership function.

$$\mu_T(x) = \begin{matrix} 1 & y \text{ in core} \\ p(x) & y \in I_1 \quad I_2 \\ 0 & \text{otherwise} \end{matrix} \qquad (4)$$

4 Spatial Indeterminacy

In the spatial indeterminacy area, [9] states that fuzziness is a property of a geographic entity. Fuzziness concerns objects that cannot be precisely defined otherwise [6]. On the other hand, uncertainty results from limitations of the observation, i.e., the measurement process [9].

4.1 Indeterminate Spatial Objects, Relationships, and Attributes

In the following, we point out the differences between spatial fuzziness and spatial uncertainty more prominently. Consider the example of the different soil zones, e.g., desert and prairie. Each zone is not precisely bound, but, rather, a *blurry* situation exists around their common boundaries. We can identify a location for which we are sure it is within the desert or the prairie, and we can find a location that is in-between. Consequently, the boundary between the two soil zones is *fuzzy*. However, for a forest divided into separate landparcels, we can clearly say what tree belongs to what landparcel. The boundaries between the land parcels are *crisp* and thus *certain*.

In contrast, let us consider the position of a moving vehicle whose location is not exactly known, e.g., a car is in New York. This example is characterized by a *lack of knowledge* about the car's location. The fact that the car is somewhere is precise. However, the lack of knowledge we have about its position introduces *uncertainty*. Without further knowledge, we can only give the probable area the car is in.

These examples indicate that the distinguishing element between fuzzy and non-fuzzy facts is a *crisp boundary*, i.e., when we cannot clearly say what belongs to what. The concept of boundary introduces the *interior/exterior* notion, i.e., what is within the boundary and what is outside. Spatial fuzziness occurs (a) in the relationships among spatial objects and (b) in spatial attributes.

On the other hand, the distinguishing element between uncertain and certain facts is *the lack of, or the error in our knowledge*, i.e., not sufficient knowledge about an otherwise precise fact. As a result, spatial uncertainty can refer to the degree of knowledge we have about an object's position. Uncertainty about an object's position leads to uncertainty about the spatial relationship among this object and its neighbors, e.g., if the exact boundary of a land parcel is not known, then, the exact relationships with its neighboring land parcels are not known either. Furthermore, uncertainty can exist for spatial attributes, when knowledge about them is limited.

4.2 Indeterminate Geometry

In this section, we examine in what ways fuzziness and uncertainty affect the concept of geometry. This is essential in defining spatial objects and spatial attributes; spatial relationships are defined in terms of the geometry of spatial objects.

Points and *regions* are the most commonly met geometries in spatial applications, while *line*, is a special case of a region. Here, we only consider simple geometries, i.e., points and regions with no holes and no disconnected parts. The following cases exist: (a) Uncertain point. A point can be crisp and uncertain, e.g., we know the approximate position of a car and can give probabilities for its location. (b) Fuzzy point. This case is not applicable, since the concepts of boundary and interior/exterior do not exist here. (c) Uncertain region. Consider the example of a landparcel with "not-exactly-known" (missing data) boundaries. (d) Fuzzy region. Since a region is determined by its boundaries (something is inside/outside, or left/right), a region can be fuzzy, e.g., consider soil zones, whose boundaries are not crisp, but transitional.

Indeterminate Points. We model *Space* as a set of points, homeomorphic to $\mathbb{N}^2$. The exact position of an object with geometry point is determinate, if it can be mapped onto a single point $p \in \mathbb{N}^2$. The position is indeterminate, if it can only be mapped to a set of points, i.e., the exact position is unknown. A probability function describes the likelihood for each point to be the position, e.g., uniform distribution tells us that there is an equal chance for each point. The probability mass function, p_x, for the indeterminate point x is

$$p_x(i) = P[x = i] : i \in \{\mathbb{N} \times \mathbb{N}\} \qquad (5)$$

where $P[x = i]$ is the probability that the position is mapped to point i, with i being a Cartesian coordinate. The probability that the position is outside the point set is 0. Further, all indeterminate positions are considered to be independent.

What applies to time points, can also be applied to indeterminate points in the spatial context; probability distributions describing positional indeterminacy can always be interpreted as fuzziness.

Indeterminate Regions. Indeterminate regions comprise uncertain and fuzzy regions. A *region* is a part of space bound by a connected set of points, the boundary. It can be determinate if the boundary points are determinate. Consequently, indeterminate points bound an indeterminate area. The following example illustrates this point.

Uncertain Regions. Consider a map made up of two discrete regions, A and B, sharing a common boundary. If we repeatedly digitize the map, assuming that our process introduces errors, we obtain a set of points that lies close to the actual boundary line. However, there will be more points closer to the actual location of the line than further away from it. Due to lack of knowledge, this distribution might take the form of a normal distribution whose mean is centered at the "true" location of the line. In Fig. 2a, we show the normal distribution of a particular boundary point. Fig. 2b shows the probability function in the continuous case.

Analogously, we can describe this uncertain region using a membership function. To determine this function that returns the grade to which an arbitrary point in space

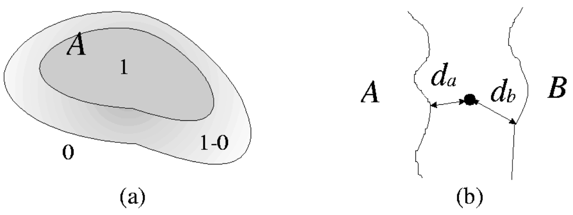


Fig. 2. Boundary point probability

belongs to an area, we split the underlying space into three parts: (a) the core of the area, (b) the boundary region, and (c) the outside. Consequently, a membership function for area *A* can be specified as follows.

$$\mu_A(i) = \begin{cases} 1 & i \in A \wedge i \notin B \\ \sum p(x) & i \in A \wedge B \\ 0 & \text{otherwise} \end{cases} \tag{6}$$

Area *B* stands for the outside of area *A* and $p(x)$ is the probability mass function of a point for being in area *A*. The argument of the membership function is a point and it returns a grade for the membership of this point in area *A*. The grade is 1 if the point is a definite member of the area and 0 if it is definitely not a member of the area. Otherwise the grade is between 1 and 0 (cf. Fig. 2a).

Fuzzy Regions. The above approach is only feasible when the probability function is known and simple, i.e., there is one probability function describing the distribution of all points in the boundary. If there were many probability functions, the membership function would become too complex to be useful. On the other hand, in some cases, we do not have "any information at all" about the boundary of a region. Consider again the transition between soil zones. The boundary exists because of the very nature of a phenomenon that is not crisp and, thus, to give a probability function describing it is not possible, or does not make sense. This illustrates the critical case for which fuzziness relieves uncertainty. We can still derive a valid membership function in assuming a smooth and steady transition from one zone to the other. A membership function for soil zones, as shown in Fig. 2b, could be characterized by the following formula (cf. [17]),

$$\mu_A(x,y) = \begin{cases} 1 & \text{if } (x,y) \in A \\ 1 - d_a / \left(d_a + d_b\right) & \text{if } (x,y) \notin A \wedge (x,y) \notin B \\ 0 & \text{otherwise} \end{cases} \tag{7}$$

where d_a and d_b are the distances from a point (*x*,*y*) to the core area of the soil zones *A* and *B*. A formula for a distance *d* from an arbitrary point given by its coordinates (*x*,*y*) to an area *A* with the boundary B_A is as follows

$$d\left((x,y), B_A\right) = \min\left\{\text{dist}\left((x,y),(m,n)\right) \mid (m,n) \in B_A\right\} \tag{8}$$

where dist(*p*,*q*) is the Euclidean distance between two points $p,q \in \mathbb{R}^2$.

Above, the assumption is that the transition between the soil zones is linear. However, the effect of other transitions on the membership function would change the formula describing the membership grade for positions outside the core.

5 Spatiotemporal Indeterminacy

After showing the nature of spatial and temporal indeterminacy as well as the way to model it, we describe the combined phenomenon, *spatiotemporal indeterminacy*. Consider the example of a moving vehicle, it is reasonable to assume that its extent does not matter in a given application, and, thus, can be reduced to point. To record its movement, we sample the object's position. We cannot answer queries about an object's movement at times in-between position samples unless we interpolate the positions, e.g., linear interpolation.

For areal objects, the change of position includes the change of their centroid and shape, which has to be interpolated as well. Consider the indeterminate region example of an island. Tides have (a) a short-term effect on its coastline, whereas (b) over a longer period of time a general drift can be observed as well. If one is only interested in the general drift, the tidal effect can be modeled as a fuzzy boundary that changes over time.

5.1 Spatiotemporal Scenarios and Indeterminate Change

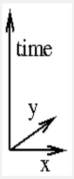
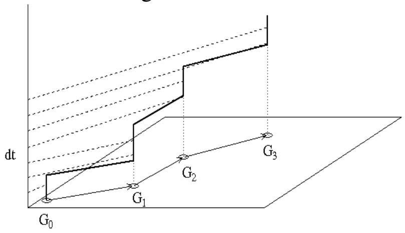
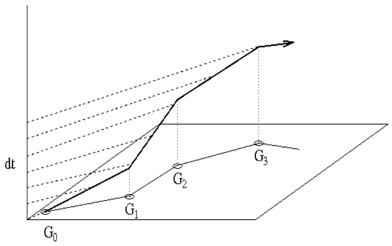
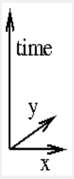
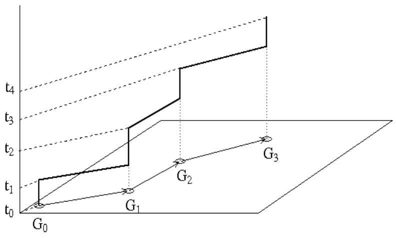
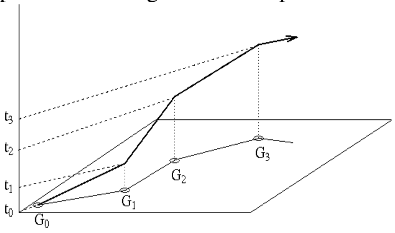
Change, or evolution, is the most important concept in the spatiotemporal context, and will in the following serve as the basis to evaluate spatiotemporal indeterminacy. As stated in literature [4], [8], [15], change (a) can either occur on a discrete or on a continuous basis and (b) can be recorded in time points or in time periods.

Table 1 illustrates the four *change* scenarios encountered in the spatiotemporal context by using a 3-dimensional representation of the temporal change of geometry. Space (x- and y-coordinates in the horizontal plane) and time (time-coordinate in the vertical direction) are combined to form a three dimensional coordinate system. In the change scenarios, the elements that can be indeterminate (with respect to an object) are *geometry*, *time point,* and *time interval*. We use a point geometry to keep the illustrations simple. However, the same change scenarios apply to other geometries. A discrete change of geometry from G_i to G_{i+1} is indicated by using an arrow in the spatial plane as opposed to a line in case of a continuous change. In the following, we examine each scenario with respect to indeterminacy.

The first case, *Scenario 1* in Table 1, is the *discrete change of a geometry recorded in time points*. Geometry stays constant for some time and then changes instantly. It is sampled at constant time intervals *dt*. The geometry and/or the time point can be indeterminate.

The second case, *Scenario 2* in Table 1, is the *continuous change of a geometry recorded in time points.* We sample a constantly changing geometry at time intervals *dt.* Knowing a geometry only at time points has two implications, (i) recording geometries at points means assessing a momentary situation without inferring anything about the geometry prior or past the time point. Consequently, (ii) time and space are

Table 1. Four spatiotemporal change scenarios

Change Time	Discrete	Continuous
Point 	**1)** Geometry is recorded at a time point. It may or may not differ from the previously recorded one. We do not know when the change occurred. 	**2)** Geometry is sampled at *time points.* In between time points we have no knowledge about the geometry. 
Period 	**3)** Geometry is valid for a given *time period.* After a change, a new time period starts. 	**4)** Geometry is sampled at time points, the starting and end points of the time period. A time period is assigned a *"change" function* that models the positional change within the period. 

independent; not knowing the exact extent of the geometry does not affect the time interval and vice versa.

In contrast, *Scenarios 3* and *4* in Table 2, suggest that a *change function* of the form $C: t_x \rightarrow G_x$ exists that determines a geometry G_x for a time point t_x in an interval spatially bound by the two geometries G_i and G_{i+1} and temporally bound by the time interval $T_i = [t_i, t_{i+1}]$. The change function C can be different for every time interval.

The third case, *Scenario 3* in Table 1, is the *discrete change of a geometry recorded in time intervals.* The objective is to "begin" a new interval when a spatial change occurs, i.e., new time intervals start at the time points t_0 through t_4. The geometry is constant within a time interval. Spatial and temporal indeterminacy affect each other. Dealing with indeterminate spatial extents, e.g., uncertainty induced by measurement errors, implies that the time point at which a change occurs cannot be detected precisely. On the other hand, having an indeterminate temporal event, e.g., clock errors, introduces spatial indeterminacy.

The last and most complex case, *Scenario 4* in Table 1, is the *continuous change of a geometry recorded in time intervals.* This case is based on the fact that for a

Table 2. Change scenarios without temporal indeterminacy

Geometry (G_i, G_{i+1})	**Time (t_i, t_{i+1})**	**Change**
Determinate	Determinate	$C : t_x \rightarrow G_x$, where G_x, depending on the change function, is determinate or indeterminate ($\tilde{G}_x$)
Indeterminate	Determinate	(a) $C : t_x \rightarrow \tilde{G}_x$, where $\tilde{G}_x$ represents a probability, $P_x(i)$, or a membership function, $\mu_x(i)$ (b) $\mu_x(i,t)$ or $P_x(i,t)$

given time interval $T_i = [t_i, t_{i+1}]$, there exists a change function that models the transformation from geometry G_i to G_{i+1}. Each of these factors, i.e., (i) the time interval, (ii) the geometry, and (iii) the change function, can be subject to indeterminacy.

In the simplest case, the geometry G_i and G_{i+1} and the time interval T_i are *determinate*, and the change function returns a determine geometry G_x for a given time point $t_x \in T_i$. Here, we assume that the change function returns the geometry coinciding with the actual movement. Is this not the case, the change function *interpolates* in between the geometries G_i to G_{i+1} and returns an indeterminate geometry. An example is to use linear interpolation, i.e., the two geometries G_i to G_{i+1} are considered to be the endpoints of a line. Section 5.2 gives an elaborate example of a change function for this case.

If we further allow G_i and G_{i+1} to be *indeterminate*, our change function would in any case return an indeterminate G_x. In the following, we use the "~" symbol on top of the parameter to denote indeterminacy. This means that if a geometry is described by a probability or membership function, this very function is subject to change in the time interval T_i.

Following the idea from before, we would have a change function that returns a probability or membership function for a given t_x (cf. Table 2(a)). However, by integrating the temporal component, we obtain a spatiotemporal probability or membership function, i.e., a function that changes with time (cf. Table 2(b)).

Until now, we always considered time to be determinate. We use time points to determine the start and the end of the current time interval T_i, and to denote the time point in question, t_x. In case t_i and t_{i+1} are indeterminate, we cannot state the beginning and the end of the time interval precisely. Thus, the association of a geometry (indeterminate or not) to a time point becomes indeterminate. However, this affects mainly the change function and can be considered in adapting its form. In considering an indeterminate time interval, we cannot, for any time point in the time interval, give

Table 3. Change scenarios incorporating temporal indeterminacy

Geometry (G_i, G_{i+1})	**Time (t_i, t_{i+1})**	**Change**
Determinate	Indeterminate	$C : \tilde{t}_x \rightarrow \tilde{G}_x$
Indeterminate	Indeterminate	(c) $C : \tilde{t}_x \rightarrow \tilde{G}_x$, where $\tilde{G}_x$ is either a probability, $P_x(i)$, or a membership function, $\mu_x(i)$ (d) $\mu_x(i,\tilde{t})$ or $P_x(i,\tilde{t})$

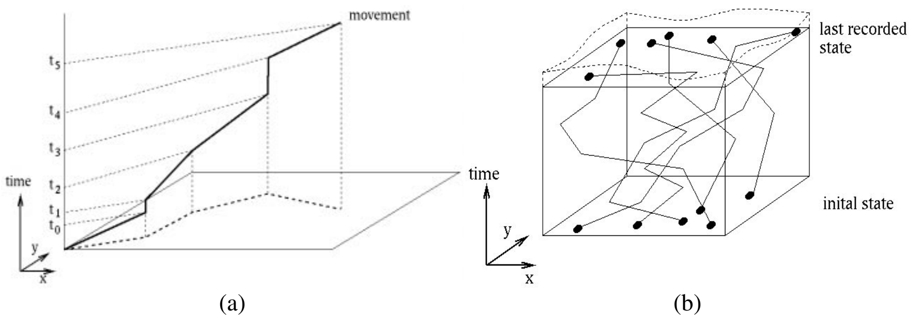


Fig. 3. Movements and space

a geometry as it would be unaffected by determinate time, but the indeterminate time contributes some additional indeterminacy. Table 3 adapts the approach shown in Table 2 to cover this case.

The central element of spatiotemporal indeterminacy is the change function manipulating geometries. This function can be seen similar to a morphing algorithm between different instances of geometries, i.e., point, line, or region. Next, we give an example illustrating the aforementioned concepts.

5.2 An Example of Use – Tracking Vehicles

Consider the application scenario in which we track the continuous movement of taxis equipped with GPS devices that transmit their positions to a central computer using either radio communication links or cellular phones.

Acquiring Movement – Sampling Moving Objects. To record the movement of an object, we would have to know the position on a continuous basis. However, practically we can only sample an object's position, i.e., obtaining the position at discrete instances of time such as every few seconds.

The solid line in Fig. 3a represents the movement of a point object. Space (x- and y-axes) and time (t-axis) are combined to form one coordinate system. The dashed line shows the projection of the movement onto two-dimensional space (x and y coordinates). A first approach to represent the movements of objects would be to store the position samples and interpolate the in-between positions. The simplest approach is to use linear interpolation. The sampled positions become the end points of line segments of polylines. The movement of an object is represented by an entire polyline in three-dimensional space. In geometrical terms, the movement of an object is termed a *trajectory* (we will use "movement" and "trajectory" interchangeably). Fig. 3b shows a spatiotemporal space (the cube in solid lines) and several trajectories (the solid lines). The top of the cube represents the time of the most recent position sample. The wavy-dotted lines symbolize the growth of the cube with time.

Measurement Error. An error can be introduced by inaccurate measurements. Using GPS measurements in sampling, the error can be described by a probability function, in our case, a bivariate normal distribution P_1.

$$P_1(x,y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}} \quad (9)$$

where σ is the standard deviation. For details on this error measure refer to [13].

Which Scenario? In Table 1 of Sect. 5.1, the sampling approach to assess the movement of objects is characterized by scenario 4. Tables 2 and 3 establish a foundation for giving a change function in between sampled position. Table 3 gives function templates in case the times of sampling are not known precisely. However, GPS allows for precise timing and, thus, we neglect the effects of time. In Table 2, Scenario 1 (determinate geometry) gives a function template in case the sampled positions are known precisely. GPS measurements are accurate but not precise. Scenario 2 (indeterminate geometry) seems to be a match for our problem. Next we show how to establish a change function to determine the position of the moving object in-between sampling. We initially assume precise position samples.

Sampling Uncertainty. Capturing the position using a GPS receiver at regular time intervals introduces *uncertainty about the position* of the object for the in-between the measurements. In this section, we give a model for the uncertainty introduced by the sampling, based on the sampling rate and the maximum speed of the object.

The uncertainty of the representation of an object's movement is affected by the *sampling rate*. This, in turn, may be set by considering the speed of the object and the desired maximum distance between consecutive samples. Let us consider the example of recording taxi movements. As a requirement, the distance between two consecutive samples should be maximally 10*m*. Given the *maximum speed* of a taxi as 150*km*/*h*, we would need to sample the position at least 4.2 times per second. If a taxi moves slower than its maximum speed, the distance between samples is less than 10*m*.

Since we did not have positional measures for the in-between position samples (cf. Fig. 4a, the object could be anywhere in between position samples), the best is to *limit the possibilities of where the moving object could have been*. Considering the trajectory in a time interval $[t_1, t_2]$, delimited by consecutive samples, we know two positions, P_1 and P_2, as well as the object's maximum speed, v_m (cf. Fig. 4b). If the object moves at maximum speed v_m from P_1 and its trajectory is a straight line, its position at time t_x will be on a circle of radius $r_1 = v_m(t_1 + t_x)$ around P_1 (the smaller dotted circle in Fig. 4b). Thus, the points on the circle represent the maximum distance of the object from P_1 at time t_x. If the object's speed is lower than v_m, or its trajectory is not a straight line, the object's position at time t_x will be somewhere within the area bound by the circle of radius r_1.

Similar assumptions can be made on the position of the moving object with respect to P_2 and t_2 to obtain a second circle of radius r_2. The constraints on the position of the moving object mean that the object can be anywhere within the intersection of the two circular areas at time t_x. This intersection is shown by the shaded area in Fig. 4b. We use the term *lens* for this area of intersection. We assume a uniform distribution

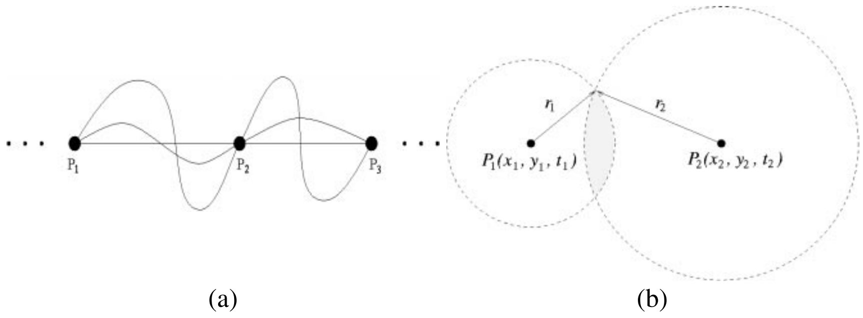


Fig. 4. (a) Possible trajectories of a moving object, (b) uncertainty between samples

for the position within the lens, i.e., the object is equally likely anywhere within this lens shape.

The sampling error at time t_x for a particular position can be described by the probability function of Equation 10, where r_1 and r_2 are the two radii described above, s is the distance between the measured positions P_1 and P_2, and A denotes the area of the intersection of the two circles.

$$P_2(x,y) = \begin{cases} 1/A & \text{for } x^2 + y^2 \le r_1^2 \wedge (x-s)^2 + y^2 \le r_2^2 \\ 0 & \text{otherwise} \end{cases} \tag{10}$$

To eliminate the radii in favor of the max speed and times, we can substitute $v_m(t_1 + t_x)$ and $v_m(t_2 - t_x)$ for the r_1 and r_2, respectively. This function describes the position of the moving object in between position samples. Thus, this function is an instance of the function template as described in Scenario 1 of Table 2.

Combining Error Sources – a Global Change Function. Table 2 gives a template of a change function that incorporates indeterminate positions. Using our example, this translates to adapting Equation 10 such that the values for x and y are not precise but affected by the measurement error. A mathematical framework suitable for this problem is *Kalman filtering* [11], which combines various error prone measurements about the same fact into a single measurement resulting in a smaller error. This mathematical framework stipulates a method to combine uncertainty to reduce the overall error. Examples of applying Kalman filtering to the domain of vehicle navigation are the integration of three independent positioning systems such as dead reckoning, map matching, and GPS, to determine the precise position of vehicles [12].

6 Conclusions and Future Work

The work presented in this paper concerns the spatial, temporal, and spatiotemporal indeterminacy, i.e., fuzzy and uncertain phenomena. We first show how the fundamental modeling concepts of *spatial objects*, *attributes*, *relationships*, *time points*, *time periods*, and *events* are influenced by *indeterminacy*. Next, we focus on the change of spatial objects and their geometry in time. We argue that change can occur on a discrete and on a continuous basis, as well as it can be recorded in time points and time periods. By combining these concepts, we present four different change scenarios, which are affected by indeterminacy to a various degree. The indeterminacy of change is formalized and combines the spatial and temporal concepts. Finally, the rather general concepts are applied to existing application areas. We discuss uncertainty existing in the context of moving-point-object applications. We give a change function to describe the position of moving objects in time, based on positional samples. The change function is influenced by measurement errors and sampling uncertainty.

Although mentioned, the paper does not discuss, directly, indeterminacy as related to relationships among spatial, temporal, or spatiotemporal objects. An extension of this work towards this direction is essential. Also, the mathematical models we presented are concrete enough to describe and motivate indeterminacy related to the temporal, spatial, and spatiotemporal domain. However, to actually implement these concepts, more detailed mathematical formulas are needed. Finally, in a more general framework, this work points towards the development of spatiotemporal data types and data structures incorporating indeterminacy.

References

1. Burrough, P.A., MacMillan, R.A., amd van Deursen, W.: Fuzzy Classification Methods For Determining Land Suitability from Soil Profile Observations and Topography. *Journal of Soil Science*, 43, pp. 193-210, 1992.
2. Cheng, T. and Molenaar, M.: Diachronic Analysis of Fuzzy Objects. *GeoInformatica* 3(4), pp. 337 - 355, 1999
3. Chrisman, N.: A Theory of Cartographic Error and Its Measurement in Digital Databases. In *Proceedings Auto-Carto 5*, pp. 159-168, 1982.
4. Claramunt, C., and Theriault, M.: Managing Time in GIS: An Event-Oriented Approach. *Recent Advances in Temporal Databases*, Springer-Verlag, pp. 142-161, 1995.
5. Dyreson, C.E., Soo, M.D., and Snodgrass, R.T.: The Data Model for Time. *The TSQL2 Temporal Query Language*, Kluwer Academics, pp. 97-101, 1995.
6. Fisher, P.: Boolean and Fuzzy Regions. *Geographic Objects with Indeterminate Boundaries*, Taylor & Francis, pp. 87-94, 1996.
7. Goodchild, M. and Gopal, S. (Eds): *Accuracy of Spatial Databases.* Taylor & Franics, 1989.
8. Güting, R., Böhlen, M., Erwig, M., Jensen, C.S., Lorentzos, N., Schneider, M., and Vazirgiannis, M.: A Foundation for Representing and Querying Moving Objects. *ACM Transactions on Database Systems* 25(1), pp. 1-42, 2000.
9. Hadzilacos, T.: On Layer-Based Systems for Undetermined Boundaries. *Geographic Objects with Indeterminate Boundaries*, Taylor & Francis, pp. 237-256, 1996.
10. Hadzilacos, T., and Tryfona, N.: A Model for Expressing Spatiotemporal Integrity Constraints. In *Proc. of the International Conference on Theories and Methods of Spatio-Temporal Reasoning*, pp. 252 - 268, 1992.
11. Kalman, R.E.: A New Approach to Linear Filtering and Prediction Problems. *Transactions of the ASME–Journal of Basic Engineering*, pp. 35-45, 1960.

12. Krakiwsky, E.J., Harris, C. B., and Wong, R.: A Kalman Filter for Integrating Dead Reckoning, Map Matching, and GPS Positioning. In *Proc. of the IEEE Position Location and Navigation Symposium*, pp. 39-46, 1988.
13. Pfoser, D. and Jensen, C.S.: Capturing the Uncertainty of Moving-Object Representations. In *Proc. of the 6th International Symposium on the Advances in Spatial Databases*, pp. 111-132, 1999.
14. Pfoser, D. and Tryfona, N.: Requirements, Definitions, and Notations for Spatiotemporal Application Environments. In *Proc. of the 6th ACM Symposium on Geographic Information Systems*, pp. 124-130, 1998.
15. Pfoser, D. and Tryfona, N.: Capturing Fuzziness and Uncertainty of Spatiotemporal Objects. *TimeCenter Technical Report*, 2001.
16. Pfoser, D., Jensen, C., and Theodoridis, Y.: Novel Approaches in Query Processing for Moving Objects Data. In *Proc. of the 27th Conference on Very Large Databases*, pp. 395-406, 2000.
17. Schneider, M., Metric Operations on Fuzzy Spatial Objects in Databases, In *Proc. of the 8th ACM Symposium on Geographic Information Systems*, 2000.
18. Shibasaki, R.: Handling Spatiotemporal Uncertainties of Geo-Objects for Dynamic Update of GIS Databases from Multi-Source Data. In *Advanced Geographic Data Modeling*, Netherlands Geodetic Commission, Publications on Geodesy, 40, pp. 228-243, 1994.
19. Vazirgiannis, M.: Uncertainty Handling in Spatial Relationships. In *Proc. of the ACM Symp. on Applied Computing*, 2000.
20. Worboys, M.: Imprecision in Finite Resolution Spatial Data. *GeoInformatica* 2(3), pp. 257-279, 1998.

Probability-Based Tile Pre-fetching and Cache Replacement Algorithms for Web Geographical Information Systems

Yong-Kyoon Kang, Ki-Chang Kim, and Yoo-Sung Kim

Department of Computer Science & Engineering
INHA University, INCHEON 402-751, Korea
yskim@inha.ac.kr

Abstract. In this paper, an effective probability-based tile pre-fetching algorithm and a collaborative cache replacement algorithm for Web geographical information systems(Web GISs) are proposed. The proposed tile pre-fetching algorithm can approximate which tiles will be used in advance based on the global tile access pattern of all users and the semantics of query so that a user request will be answered quickly since the needed tiles are likely in cache database. When a client runs out of cache space for newly down-loaded tiles, the proposed cache replacement algorithm determines which tiles should be replaced based on the future access probabilities. By combining the proposed tile pre-fetching algorithm with the cache replacement algorithm, the response time for user requests can be improved substantially in Web GIS systems.

1 Introduction

With the rapid growth of computer hardware and software technologies and the user's requirements for geographical information, geographical information systems (GISs) that can analyze, process, and manage geo-spatial data have been developed and become very popular in several fields, e.g. civil engineering and computer engineering ([1]). Furthermore, since the Internet and World Wide Web(WWW) have become very popular in real worlds, users can get geographical information at a low cost from the Web servers that can provide geographical information. These systems are referred to as Web GIS systems ([2,3,4,5,6]).

The types of Web GIS systems can be classified into server-side Web GIS and client-side Web. In server-side Web GIS systems, since the server has to process all requests of all clients, the server might be over-loaded, and the response time for user requests may become too slow ([2,7]). On the other hand, in client-side Web GIS systems, the client loads the geographical data processing modules from the server when it makes a connection to the server. From then on, the client can process users' requests by itself. Recently, client-side Web GIS systems have become very popular and several real systems are being developed.

The granularity of transmission from the server to the client can be either a whole map or a tile of the map ([2]). If the granularity is a whole map, the server searches all spatial objects and all aspatial information on GIS database to retrieve relevant objects and information and sends the map with the retrieved aspatial information to the client. However, a map may be of very large size, since a map can include a large number of objects. It is clear that the larger the size of a communication unit becomes, the more loading time is needed between the server and the client. To reduce the initial loading time, many systems have adopted the concepts of tiling and layering. Tiling divides the map into several small pieces so that each of them can be transferred in a short time, while layering partitions the map into several layers such that each layer represents some specific information.

Tiling can minimize the initial user's response time, but it alone can't minimize the total response time. To minimize the total response time, the system should pre-fetch some tiles that are likely to be accessed in advance and save them in a cached database for future reusing. When the user requests the tiles that have been pre-fetched and saved at the cached database, the client can give these tiles to the user without the communication delay to fetch the required tiles from the server.

In this paper, an efficient tile pre-fetching algorithm for Web GIS based on users' global access pattern is proposed. In the proposed algorithm, the server collects and maintains the transition probabilities between adjacent tiles. With these probabilities the server can predict which tiles have the higher probability of accessing in next time than others, and by pre-fetching those recommended tiles, the client can respond to the user's requests much faster.

When the client's cache is run out of space, the client should determine which tiles to replace with newly fetched tiles. Those tiles that are not likely to be accessed in the near future can be replaced, and the client should be able to select such tiles. We propose a cache replacement algorithm that predicts the future usage of the tiles correctly, based on the same access probabilities that are calculated and used for tile pre-fetching. The proposed cache replacement algorithm selects tiles with small transition probabilities from the current requested tile as candidate tiles for replacement.

The rest of this paper is organized as following. In Sect. 2, we discuss the architecture of Web GIS systems and describe query processing in it. In Sect. 3, we propose an efficient tile pre-fetching algorithm that can determine a set of tiles that are likely to be requested in the near future, based on the global tile access patterns and a cache replacement algorithm that can collaborate with the proposed tile pre-fetching algorithm in Web GIS systems. We also discuss an example that can show the effectiveness of the proposed tile pre-fetching algorithm and cache replacement algorithm. Finally, we conclude the paper in Sect. 4.

2 Query Processing in Web GIS Systems

The general architecture of Web GIS systems in which the proposed tile pre-fetching and cache replacement algorithms are used is in Fig. 1 ([2]). In Fig. 1, we do not include all components of Web GIS systems. That is, Fig. 1 shows the abstracted architecture of Web GIS systems.

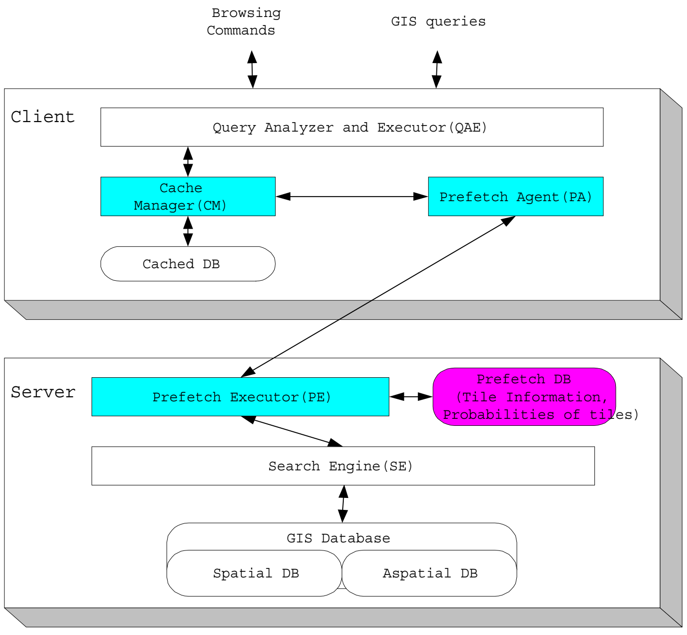


Fig. 1. Abstracted architecture of Web GIS systems

A Web GIS system mainly consists of two components; clients and a server. A client is the Web browser with several data processing facilities that are loaded from the server when the client makes a connection to the server. Server manages the GIS database that consists of spatial database and aspatial database and provides useful information to the clients when the clients submit user's requests. As we discussed in the previous section, the server manages the spatial information in the unit of tile. That is, a map is decomposed into a set of tiles that can be transferred to the client in a short time.

In Web GIS systems, users' requests are classified into two categories; browsing commands and GIS queries. GIS queries include point queries, region queries, and object retrieval queries with selection predicates. To use the point query, the user gives the coordinates (a, b) of a point in a map, and the client serves the tile that has the given point. To use the region query, the user gives a specified region in rectangular form with (a_1, b_1) and (a_2, b_2) or circular form with the center point (a, b) and a radius, and the client returns a set of tiles that covers the given region. For an object retrieval query with selection predicates, the client returns the objects that satisfy the given predicates on the map. Usually, GIS queries such as point queries, region queries, and object retrieval queries are used as the first request when users make a session to the Web GIS systems. That is, the user first submits a GIS query and, based on the result of the first GIS query, submits a sequence of browsing commands and/or GIS queries. Browsing commands include zooming and moving commands. To execute a zoom-in command, by which the user can see the current position (a, b) of the map in more detail, if the required data has been cached at client, the client doesn't have to go to the server. Otherwise, however, client should down-load more detailed map information from the server. For a zoom-out command, by which the user can view the map in wider area, the client should fetch neighbor tiles of the current one from the server. By using the moving commands, users can retrieve 4 neighbor tiles of the current tile. That is, by using a moving command, users can move to one of the 4 neighbor tiles in the direction of up, down, left, and right, respectively.

The formats of users' requests in Web GIS systems are as following.

```
Point_Query(a, b)
Rectangle_Region_Query (a1, b1, a2, b2)
Circle_Region_Query(a, b, radius)
Objet_Retireval_Query(selection_predicates)
Zoom-in(a, b, smaller_radius)
Zoom-out(a, b, larger_radius)
Moving(a, b, direction)
```

In Fig. 1, user's request is processed as following. When a user submits a request to the client. Query Analyzer and Executor (QAE) analyzes the user's request and executes it. To execute user's request, QAE requests necessary data that should be processed for user's request from Cache Manager (CM). If CM can find the data in the cached database, it transfers the data without requesting to the server. Otherwise, CM sends the request to Pre-fetch Agent (PA) to retrieve the necessary data from the server. PA basically tosses the user's request to Pre-fetch Executor (PE) at the server. However, to give some information needed for tile pre-fetching to the server, PA adds some additional information to the original users' requests. That is, in addition to the original request, PA also gives the list of tiles that have been cached, the transition frequencies among the cached tiles since the last connection to the server, and the number of tiles it wants to pre-fetch. The pre-fetching size can be determined on the basis of the size of free space for tile pre-fetching, and the regularity degree of access pattern of the user at the client.

When PE receives a modified request from the PA, PE decomposes the request into the original request and the augmented information. It sends the original request to Search Engine (SE) to retrieve the result of the user's request and performs pre-fetching based on the retrieved result and the augmented information. To properly pre-fetch tiles that are likely to be accessed by the user in the near future, the pre-fetching algorithm uses the transition probabilities between tiles, and the details on the pre-fetching algorithm will be discussed in Sect. 3.1.

PE sends the pre-fetching result tiles with the retrieved tiles for a user's request to CM through PA. When CM receives too many tiles than its storage capacity, CM determines which space will be replaced with the newly received results by using the cache replacement algorithm described in Sect. 3.2.

When QAE receives the retrieved result, QAE executes the user's request on the retrieved data, and the final result is shown to the user's browser.

3 Tile Pre-fetching and Cache Replacement in Web GIS Systems

3.1 A Probability-Based Tile Pre-fetching Algorithm

PE determines which tiles should be pre-fetched based on the updated global access pattern information according to Algorithm 1. To determine tiles to be pre-fetched, PE first updates the global access pattern by using the local access pattern sent from PA (step 1). If the number of tiles returned as the result of the request is greater than pre-fetch_size, we do not need to pre-fetch, since the request has retrieved more tiles than the expectation specified by pre-fetch_size (step 2). Otherwise, we calculate the normalized probabilities from $T_{x,y}$ to its 4 neighbor tiles that are within distance = 1 (step 3). If the pre-fetch size is greater than 1, we calculate the transition probabilities to those tiles located within distance ≤ pre-fetch_size (step 4~5). At step 6, we sort the probabilities in descending order. Then we select top-ranked pre-fetch_size tiles from the pre-fetching space (step 7). At step 8, we eliminate the tiles that have been already cached at the client by the previous requests. The list of tiles to be pre-fetched is returned as the result of Algorithm 1. And, Algorithm 1 also returns the own_tile_list with the updated transition probabilities, and the result is sent to the CM of client for cache replacement.

```
Algorithm 1: Pre-fetching Algorithm Based on Global Access Pattern

Input: pre-fetch_size, own_tile_list, local_access_pattern, return_tiles including
    central tile $T_{x,y}$ with specified point (a, b)
Output: list of tiles with transition probabilities to be pre-fetched for pre-fetching,
    own_tile_list with the updated transition probabilities for cache replacement
Data Structure: transition probability matrix


1:  Updates the global access pattern by using local_access_pattern;
2:  IF (number of return_tiles > pre-fetch_size) RETURN (NO_PRE-FETCH);
3:  Computes the normalized probabilities from $T_{x,y}$ to its 4 neighbors /*distance
    = 1*/;
4:  FOR each tile within distance from 2 to pre-fetch_size DO /* distance ≥ 2 */
5:  Compute the conditional probability of tile moving from $T_{x,y}$ to the tile;
6:  Sorts the probabilities of tiles within the pre-fetching space of distance ≤ pre-
    fetch_size;
7:  Let pre-fetch_list = select top-ranked pre-fetch_size tiles within the pre-
    fetching space;
8:  Let pre-fetch_list = pre-fetch_list – own_tile_list;
9:  Resets the transition probabilities of all tiles in {own_tile_list – pre-
    fetch_list} to 0;
10: RETURN(pre-fetch_list and own_tile_list with the updated transition prob-
    abilities);
```

As an example we consider a pre-fetching request that returns tile $T_{x,y}$[1] as the retrieval result. If pre-fetch_size is 0, Algorithm 1 quits with NO_PRE-FETCH at step 2 after updating the global access pattern by using the local_access_pattern. Otherwise, i.e., if pre-fetch_size is greater than 0, step 3 of Algorithm 1 is executed. Assume that pre-fetch_size is 3, which means PA want to pre-fetch up to 3 tiles. PE forms the pre-fetching space with the distance equal to 3. The pre-fetching space includes tiles that

[1] In general, *Rectangle_Region_Query, Circle_Region_Query, Object_Retrieval_Query*, and *Zoom-out* commands might return more than one tile as the retrieved result. For simplicity, however, we assume that a single tile, $T_{x,y}$, is retrieved.

can be accessed within the specified number of tile movements from the retrieved tile, $T_{x,y}$. Fig. 2 shows an example of pre-fetching space from $T_{x,y}$ with pre-fetch_size= 3. Within the pre-fetching space, the four immediate neighbor tiles of $T_{x,y}$, $T_{x,y+1}$, $T_{x+1,y}$, $T_{x,y-1}$, and $T_{x-1,y}$ can be reached by 1 tile movement from $T_{x,y}$. Tiles $T_{x,y+2}$, $T_{x+1,y+1}$, $T_{x+2,y}$, $T_{x+1,y-1}$, $T_{x,y-2}$, $T_{x-1,y-1}$, $T_{x-2,y}$, and $T_{x-1,y+1}$ can be reached by 2 tile movements from $T_{x,y}$. Finally, tiles $T_{x,y+3}$, $T_{x+1,y+2}$, $T_{x+2,y+1}$, $T_{x+3,y}$, $T_{x+2,y-1}$, $T_{x+1,y-2}$, $T_{x,y-3}$, $T_{x-1,y-2}$, $T_{x-2,y-1}$, $T_{x-3,y}$, $T_{x-1,y+2}$, and $T_{x-1,y+1}$ can be reached by 3 tile movements from $T_{x,y}$. In Fig. 2, the edge from a tile to a neighbor tile stands for a tile movement, and the label of the edge means the probability for such a transition. That is, P(x,y→x,y+1) stands for the probability of tile moving from $T_{x,y}$ to $T_{x,y+1}$.

For tiles that can be reached by 1 tile movement from $T_{x,y}$, we compute the normalized probabilities. The normalization of the probabilities is necessary because a specified position (a, b) in $T_{x,y}$ has an effect to the next tile movement. That is, if the specified point (a, b) is near to the upper border, then the user who has specified the point is likely to move to the upper tile than the lower one, and so on.

To explain the normalization process, let's consider the situation depicted in Fig. 3. The original probabilities of tile moving from $T_{x,y}$ to $T_{x,y+1}$, $T_{x+1,y}$, $T_{x,y-1}$, and $T_{x-1,y}$ are P(x,y→x,y+1), P(x,y→x+1,y), P(x,y→x,y-1), and P(x,y→x-1,y), respectively. The specified location by the user is (a,b). Let's represent the normalized transition probabilities with distance 1 from $T_{x,y}$ as P'(x,y→x,y+1), P'(x,y→x+1,y), P'(x,y→x,y-1), and P'(x,y→x-1,y), respectively. Note that the summation of the normalized probabilities, P'(x,y→x+1,y) and P'(x,y→x-1,y) along the x axis should be same as the summation of the original probabilities of P(x,y→x+1,y) and P(x,y→x-1,y), and they should reflect the internal division ratio of the specified position along the x axis. Equations (1) and (2) show the formula for P'(x,y→x+1,y) and P'(x,y→x-1,y), respectively. In equations (1) and (2), for simplicity, we use P_{right} and P_{left} instead of P(x,y→x+1,y) and P(x,y→x-1,y), respectively. A similar argument can be made for P'(x,y→x,y+1) and P'(x,y→x,y-1), and the resulting formulas are in equations (3) and (4), respectively. In equations (3) and (4), again for simplicity, we use P_{up} and P_{down} instead of P(x,y→x,y+1) and P(x,y→x,y-1), respectively.

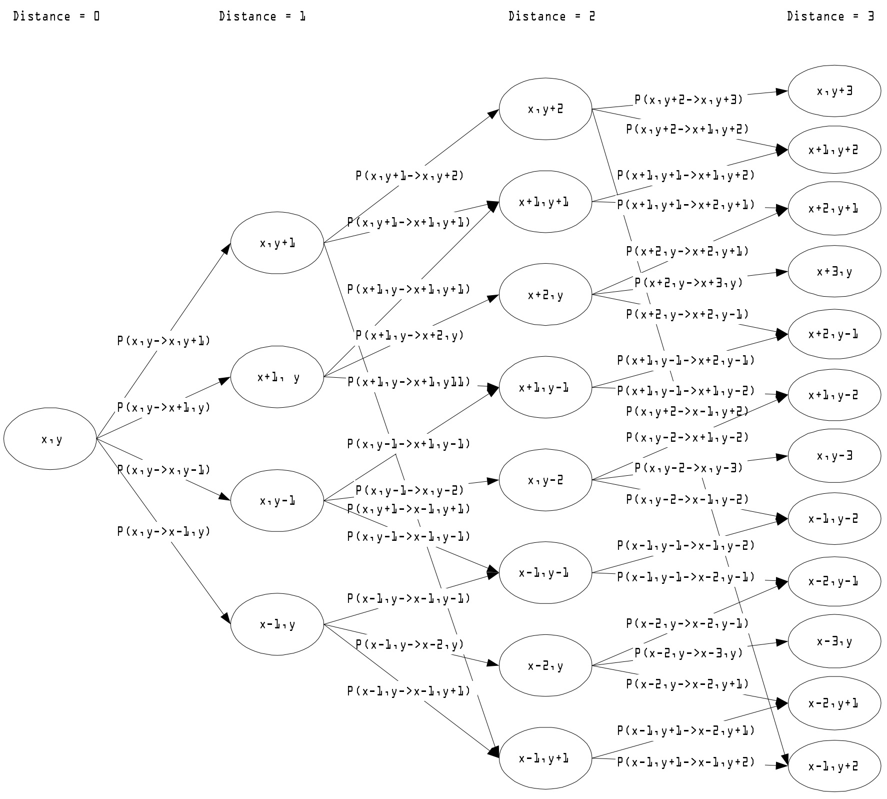


Fig. 2. Pre-fetching space within distance ≤ 3 from $T_{x,y}$

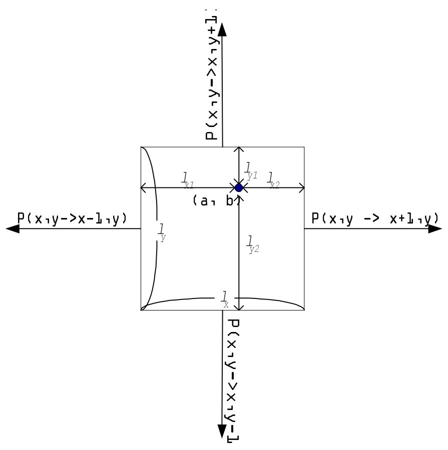


Fig. 3. Normalized probabilities from $T_{x,y}$ to Its 4 neighbor tiles

$$P'_{right} = \frac{(P_{right} + P_{left})}{(l_{x1}P_{right} + l_{x2}P_{left})} \times l_{x1}P_{right} \tag{1}$$

$$P'_{left} = \frac{(P_{right} + P_{left})}{(l_{x1}P_{right} + l_{x2}P_{left})} \times l_{x2}P_{left} \tag{2}$$

$$P'_{up} = \frac{(P_{up} + P_{down})}{(l_{y2}P_{up} + l_{y1}P_{down})} \times l_{y2}P_{up} \tag{3}$$

$$P'_{down} = \frac{(P_{up} + P_{down})}{(l_{y2}P_{up} + l_{y1}P_{down})} \times l_{y1}P_{down} \tag{4}$$

To compute the probability of tile moving from $T_{x,y}$ to a tile that is within distance = 2, we can use the conditional probability computation. As an example, let's consider how to compute the conditional probability of tile moving from $T_{x,y}$ to the tile $T_{x,y+2}$. From Fig. 2, we can see that $T_{x,y+2}$ can be reached by two tile movements from $T_{x,y}$ through $T_{x,y+1}$. That is, we can reach to $T_{x,y+2}$ from $T_{x,y}$ by moving first to $T_{x,y+1}$ and then moving to $T_{x,y+2}$. Thus, we can compute the conditional probability of tile moving from $T_{x,y}$ to $T_{x,y+2}$ through $T_{x,y+1}$ by equation (5).

$$P(x,y \to x,y+2) = P'(x,y \to x,y+1) \times P(x,y+1 \to x,y+2) \tag{5}$$

Some tiles can be reached in several ways. For example, $T_{x+1,y+1}$ can be reached from $T_{x,y}$ by using two different paths (see Fig. 2): one is $T_{x,y} \to T_{x,y+1} \to T_{x+1,y+1}$, and the other is $T_{x,y} \to T_{x+1,y} \to T_{x+1,y+1}$. In this case, the conditional probability of tile moving from $T_{x,y}$ to $T_{x+1,y+1}$ can be computed as follows.

$$\begin{aligned} P(x,y \to x+1,y+1) &= P'(x,y \to x,y+1) \times P(x,y+1 \to x+1,y+1) \\ &+ P'(x,y \to x+1,y) \times P(x+1,y \to x+1,y+1) \end{aligned} \tag{6}$$

We can do the similar computation for the tiles of distance = 3. The conditional probability of tile moving from $T_{x,y}$ to $T_{x,y+3}$ is computed by using equation (7) (see also Fig. 2).

$$P(x, y \rightarrow x, y+3) = P(x, y \rightarrow x, y+2) \times P(x, y+2 \rightarrow x, y+3) \tag{7}$$

Also, for $T_{x+1,y+2}$ in Fig. 2, the conditional probability of tile moving from $T_{x,y}$ to $T_{x+1,y+2}$ is computed by using equation (8).

$$P(x, y \rightarrow x+1, y+2) = P(x, y \rightarrow x, y+2) \times P(x, y+2 \rightarrow x, y+3) \tag{8}$$

$$+ \; P(x, y \rightarrow x+1, y+1) \times P(x+1, y+1 \rightarrow x+1, y+2)$$

Generally, the conditional probabilities of tile moving from $T_{x,y}$ to $T_{x+n,y+m}$, where the maximum distance is |n| + |m|, can be computed as in equation (9).

$$P(x,y \rightarrow x+n,\ y+m) = SUM(conditional\ probabilities\ of\ all\ paths\ from\ T_{x,y}\ to\ T_{x+n,y+m}) \tag{9}$$

After computing the conditional probabilities to all tiles within distance ≤ pre-fetch_size from the retrieved tile, $T_{x,y}$, at step 4~5 of Algorithm 1, the list of tiles that should be pre-fetched is selected according to step 6~8. To remove the tiles that have been saved in the client's cache database but will not be used in future from the cache database for making free cache space for the current request, step 9 of Algorithm 1 resets the transition probabilities of these tiles to 0. Then, the cache space for these tiles can be replaced when CM needs more space for the newly fetched tiles. As the result of Algorithm 1, the list of tiles to be pre-fetched and the own_tile_list with the updated transition probabilities are returned.

After PE pre-fetches the tiles of pre-fetch_list, it returns the retrieved result tiles that are actually retrieved for the request, the pre-fetched tiles with the transition probabilities, and the own_tile_list with the updated transition probabilities to CM through PA of the client issuing the request.

3.2 A Collaborative Cache Replacement Algorithm

When CM of a client receives the result described above for a user request from PE, CM stores both the retrieved tiles and the pre-fetched tiles in the cache. However, when it runs out of the cache space, it should remove some tiles to prepare free space for newly fetched tiles. To determine which tiles should be removed, the proposed cache replacement algorithm (Algorithm 2) utilizes the transition probabilities for tiles in own_tile_list, which are already computed for the purpose of tile pre-fetching (Algorithm 1). Algorithm 2 selects a set of tiles that have relatively smaller values of transition probability from the current position than others among own_tile_list as the victim for cache replacement.

```
Algorithm 2: Cache Replacement Algorithm

Input: retrieved result tiles, pre-fetched tiles, and own_tile_list with the transition
      probabilities
Data structure: list of cached tiles, size of free cache space

1:   victim_tile_list = NULL;
2:   required space = retrieved result tiles + pre-fetched tiles;
3:   WHILE (size_of required space > size of free cache space ) DO {
4:   select tile $T_{i,j}$ that has the minimum transition probability from own_tile_list;
5:   victim_tile_list + = { $T_{i,j}$ };
6:   own_tile_list – = { $T_{i,j}$ };
7:   size of free cache space + = size_of($T_{i,j}$); } /* for making enough space */
8:   remove tiles in victim_tile_list from the cached database;
9:   list of cached tiles – = victim_tile_list;
10:  saves retrieved result tiles and pre-fetched tiles into the cached database;
11:  list of cached tiles += (retrieved result tiles + pre-fetched tiles);
12:  size of free cache space – = size_of(retrieved result tiles + pre-fetched tiles);
13:  RETURN;
```

In Algorithm 2, step 3 through step 7 select the set of tiles that should be removed from the cached database to make enough free space for storing the retrieved result tiles and the pre-fetched tiles. The selected tiles are actually removed from the cached database (step 8) and the list of cached tiles (step 9). At step 10, the retrieved tiles and the pre-fetched tiles are stored in the cached database. After cache replacement, the list of actual cached tiles and the size of free cache space are updated at step 11 and 12, respectively.

3.3 Effects of the Proposed Tile Pre-fetching and Cache Replacement Algorithms

To show the effectiveness of the proposed tile pre-fetching and cache replacement algorithms, we discuss an example in this subsection. Assume that all tiles are in same size and the cached database can store 5 tiles at maximum and PA submits the following query to PE.

```
Point_Query(a, b, pre-fetch_size(=2)) with
own_tile_list(={(x,y-1),(x+1,y-1),(x,y-2),(x-1,y-1)}),
local_access_pattern(=NULL).
```

And as the result of the above query, assume tile $T_{x,y}$ is returned. The specified point (a, b) divides internally the horizontal length of $T_{x,y}$ into the ratio of 6:4 for $l_{x1}:l_{x2}$ of Fig. 3. Also the specified point (a, b) divides internally the vertical length of $T_{x,y}$ into the ratio of 1:9 for $l_{y1}:l_{y2}$ of Fig. 3. We also assume that the original probabilities for tile moving from a tile to its neighbor tiles are as shown in Fig. 4.

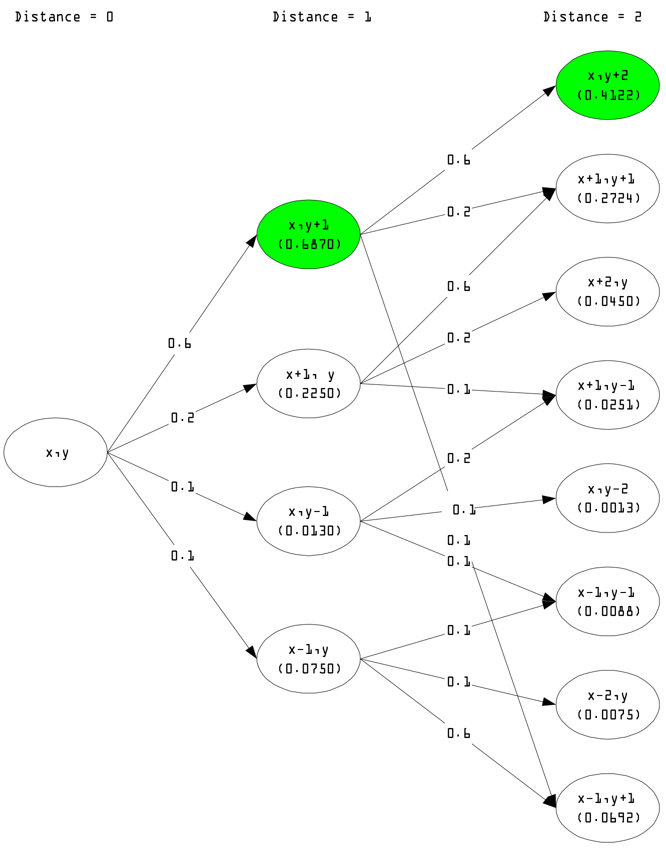


Fig. 4. A pre-fetching space within distance ≤ 2 from $T_{x,y}$

According to Algorithm 1, the global access pattern is first updated by using the local access pattern. However, since the local access pattern is NULL, the global access pattern does not change. Since the number of tiles returned as the result of the query is

1, and pre-fetch_size(=2) is greater than the number of returned tiles(=1), step 3 of Algorithm 1 computes the normalized probabilities of tile moving from $T_{x,y}$ to its 4 neighbor tiles by using equations (1), (2), (3), and (4), respectively. The normalized probabilities are shown within parentheses at the four nodes of distance = 1 in Fig. 4. And, the conditional probabilities for the tiles of distance = 2 is computed by using either equation (5) or equation (6) according to the number of incoming branches into the node in Fig. 4, and the computed result is also shown at each node of distance = 2 with parentheses in Fig. 4.

Among the nodes in the pre-fetching space of distance ≤ pre-fetch_size(=2), the top-ranked 2 tiles are selected for the pre-fetching list. Hence, $T_{x,y+1}$ and $T_{x,y+2}$ are selected for actual pre-fetching. The final result for the above *Point_Query*, therefore, is $T_{x,y}$, $T_{x,y+1}$, and $T_{x,y+2}$. $T_{x,y}$ is transferred to the client as the actual retrieved result of the query, and $T_{x,y+1}$ and $T_{x,y+1}$ are also transferred to the client as the pre-fetching results.

When CM of the client issuing the query receives the result, $T_{x,y}$, $T_{x,y+1}$, and $T_{x,y+2}$, it runs out of free space for caching because 4 tiles, $T_{x,y-1}$, $T_{x+1,y-1}$, $T_{x,y-2}$, and $T_{x-1,y-1}$ have been stored at its cached database in which 5 tiles can be stored at maximum, i.e., the size of free cache space is 1. Hence, as the victims for cache replacement, Algorithm 2 selects $T_{x,y-2}$ and $T_{x-1,y-1}$ among the own_tile_list since these have smaller transition probabilities than others. Finally, the cached database has stored $T_{x,y}$, $T_{x,y+1}$, $T_{x,y+2}$, $T_{x,y-1}$, and $T_{x+1,y-1}$ after the complete execution of the above *Point_Query*.

In that case, as long as user moves around these tiles, the communication between the client and the server is not needed since the client can serve these user's requests without down loading the additional tiles from the server. However, if these tiles have not been pre-fetched, the client should down load these tiles newly, and the user has to wait until these tiles are completely fetched into the cached database. So, by using the proposed pre-fetching algorithm and the collaborative cache replacement algorithms, the response time can be remarkably improved in Web GIS systems.

4 Conclusions

In this paper, we have proposed an effective tile pre-fetching algorithm that is able to determine which tiles are likely to be accessed in the near future according to the global access pattern of all users in Web geographical information systems(Web GISs). And we have also proposed a collaborative cache replacement algorithm that can work with the proposed tile pre-fetching algorithm. The proposed cache replacement algorithm determines which tile space should be removed from the client's cached database based on the transition probabilities already computed for tile pre-fetching. We have modified the architecture of Web GIS systems to accommodate the proposed pre-fetching algorithm with the collaborative cache replacement algorithm and showed that the proposed algorithms improved the response time substantially.

As the future works, we are doing the experimentation to inspect the performance of the proposed pre-fetching and cache replacement algorithms through simulation. We also plan to make an adaptation of the proposed algorithms into a Web GIS engine.

Acknowledgement. This work was supported by a grant from INHA University.

References

1. R. Laurini and D. Thompson, “Fundamentals of Spatial Information Systems”, ACADEMIC Press, 1992.
2. Young-Sub Cho, A Client-side Web GIS Using Tiling Storage Structure and Hybrid Spatial Query Processing Strategy, Ph. D. Thesis, Dept. of Computer Science and Engineering, INHA University, 1999.
3. Edwardm P. F. Chan and Koji Ueda, “Efficient Query Result Retrieval over the Web”, The Proceedings of 7th International Conference on Parallel and Distributed Systems (ICPADS 00), July 2000. Page 161-170.
4. K. E. Foote and A. P. Kirvan, “WebGIS, NCGIA Core Curriculum in GIScience”, http://www.ncgia.uscb.edu/giscc/units/u133/u133.html, December 1997.
5. Serena Coetzee and Judith Bishop, “A New Way to Query GISs on the Web”, IEEE Software, May/June 1998.
6. M. V. Liedekerke, A. Jones, and G. Graziani, “The European Tracer Experiment Information System: Where GIS and WWW meet”, The Proceedings of the 1995 ESRI user Conference, http://www.esri.com/library/userconf/proce95/to050/p022.html
7. Y. K. Choo and C. Lee, “Integrated Distributed Geographical Information System (IDGIS)”, The Proceeding of the 1997 ESRI User Conference, http://www.esri.com/library/userconf/proc97/TO150/PAP101/P101.HTM
8. D. J. DeWitt, N. Kabra, J. Luo, J. M. Patel, and J. B. Yu, “Client-Server Paradise”, The Proceedings of the 20th VLDB Conference, 1994.
9. M. Carey, M. Franlin, Ml Livny, and E. Schekita, “Data Caching Tradeoffs in Client-Server DBMS Architecture”, The Proceedings of the ACM SIGMOD, Vol. 20, 1991.

Optimizing Pattern Queries for Web Access Logs[⋆]

Tadeusz Morzy, Marek Wojciechowski, and Maciej Zakrzewicz

Poznan University of Technology
Institute of Computing Science
ul. Piotrowo 3a, 60-965 Poznan, Poland
Tadeusz.Morzy@put.poznan.pl
Marek.Wojciechowski@cs.put.poznan.pl
Maciej.Zakrzewicz@cs.put.poznan.pl

Abstract. Web access logs, usually stored in relational databases, are commonly used for various data mining and data analysis tasks. The tasks typically consist in searching the web access logs for event sequences that support a given sequential pattern. For large data volumes, this type of searching is extremely time consuming and is not well optimized by traditional indexing techniques. In this paper we present a new index structure to optimize pattern search queries on web access logs. We focus on its physical structure, maintenance and performance issues.

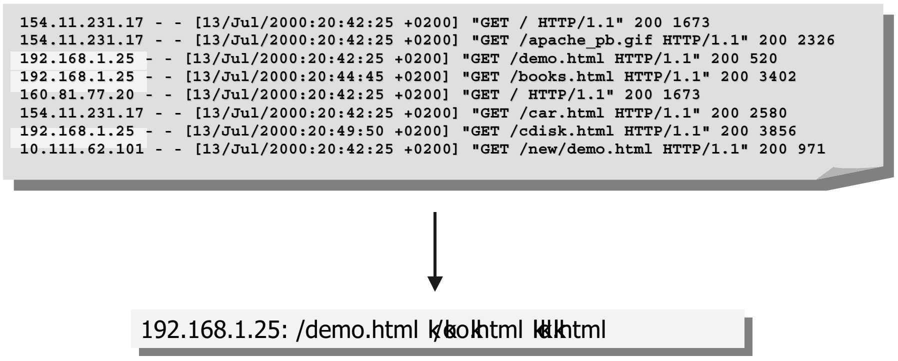
1 Introduction

Web access logs represent the history (the sequences) of users' visits to a web server [11]. Log entries are collected automatically and can be used by administrators for web usage analysis [4][6][7][16][17][18][20]. Usually, after some frequently occurring sequential patterns are discovered [1], the logs are searched for access sequences that contain (support) the discovered sequential patterns. We will refer to this type of searching as to *pattern queries*.

Example web access log is shown in Fig. 1. For each client's request we store the client's IP address, the timestamp, and the URL address of the requested object. In general, several requests from the same client may have identical timestamps since they can represent components of a single web page (e.g. attached images). In most cases, web access logs are stored in relational, *SQL*-accessed databases. Let us consider the following example of using the relational approach to pattern queries. Assume that the relation *R(IP, TS, URL)* stores web access sequences. Each tuple contains the sequence identifier (*IP*), the timestamp (*TS*), and the item (*URL*). Our example relation R describes three web access sequences: $\{A, B\} \rightarrow \{C\} \rightarrow \{D\}$, $\{A\} \rightarrow \{E, C\} \rightarrow \{F\}$, and $\{B, C, D\} \rightarrow \{A\}$. Let the searched sequential pattern (subsequence) be: $\{A\} \rightarrow \{E\} \rightarrow \{F\}$. We are looking for all the web access sequences that contain the given sequential

[⋆] This work was partially supported by the grant no. KBN 43-1309 from the State Committee for Scientific Research (KBN), Poland.

pattern. Fig. 2 gives the relation R and the SQL query, which implements the pattern query.



```
154.11.231.17 - - [13/Jul/2000:20:42:25 +0200] "GET / HTTP/1.1" 200 1673
154.11.231.17 - - [13/Jul/2000:20:42:25 +0200] "GET /apache_pb.gif HTTP/1.1" 200 2326
192.168.1.25 - - [13/Jul/2000:20:42:25 +0200] "GET /demo.html HTTP/1.1" 200 520
192.168.1.25 - - [13/Jul/2000:20:44:45 +0200] "GET /books.html HTTP/1.1" 200 3402
160.81.77.20 - - [13/Jul/2000:20:42:25 +0200] "GET / HTTP/1.1" 200 1673
154.11.231.17 - - [13/Jul/2000:20:42:25 +0200] "GET /car.html HTTP/1.1" 200 2580
192.168.1.25 - - [13/Jul/2000:20:49:50 +0200] "GET /cdisk.html HTTP/1.1" 200 3856
10.111.62.101 - - [13/Jul/2000:20:42:25 +0200] "GET /new/demo.html HTTP/1.1" 200 971
```

```
192.168.1.25: /demo.html /books.html /cdisk.html
```

Fig. 1. Web access log example and a web access sequence

IP	TS	URL
1	1	A
1	1	B
1	2	C
1	3	D
2	1	A
2	2	E
2	2	C
2	3	F
3	1	B
3	1	C
3	1	D
3	2	A

```
SELECT IP
FROM   R R1, R R2, R R3
WHERE  R1.IP=R2.IP
  AND  R2.IP=R3.IP
  AND  R1.TS<R2.TS
  AND  R2.TS<R3.TS
  AND  R1.URL='A'
  AND  R2.URL='E'
  AND  R3.URL='F';
```

Fig. 2. The relation of web access sequences and the pattern query

Since web access logs tend to be very large, there is a problem of appropriate optimizing the database access while performing pattern queries, e.g. by means of the above SQL query. Database research has developed many *indexing techniques*, like B^+-trees [5], bitmapped indexes [15], k-d-trees [3], R-trees [10], which are used to optimize queries based on exact matches of single tuples. However, these techniques do not significantly improve pattern queries, which deal with partial matches of multi-tuple sequences. There are also proposals for set-based indexing [8][14], which is used to improve subset searching (e.g. find all papers containing "data mining" and "data warehousing" in a keyword list). However, these methods work for retrieval of unordered sets of items only.

In order to realize the shortcomings of the existing indexing methods, let us consider applying B^+-tree and set-based indexes to execute the query from Fig. 2:

1. Using a B^+-tree index, tuples containing all items of each web access sequence are joined first (by IP attribute), and then the verification is done whether they contain the given items in the given order. This approach can be fairly ineffective since a web access sequence may span across many disk block, what results in multiple scanning of each block of the relation.

2. Using a set-based index, the sequence identifiers (*IP* attribute) of all sequences, which contain the searched items in any order, are found, and then the sequences are read from the relation (perhaps with help of a B^+-tree) to verify the ordering of their items. This approach gives much better results, as compared to a B^+-tree index, however, the significant overhead comes from reading and verifying the sequences having incorrect ordering.

In this paper we consider pattern queries on web access log databases. Such databases are characterized by relatively small number of items (URLs), which occur frequently in various order, and therefore a set-based index is not efficient. We present a new bitmap-oriented indexing method, which optimizes the problem of pattern queries. The basic idea behind our method, as compared to set-based indexes, is that the index structure includes not only the items of a sequence, but also the ordering of the items. In this way, we reduce the number of web access sequences needlessly read from the database, what results in shorter query execution time. We performed several experiments, which showed the significant improvement over existing indexing methods.

The structure of the paper is as follows. Section 2 describes the sequential index structure and algorithms to create and to use the index. In Sect. 3 we present the results of our performance experiments. Section 4 contains final conclusions.

1.1 Basic Definitions and Problem Formulation

Let $L = l_1, l_2, ..., l_k$ be a set of literals called items (URLs). *Web access sequence* $S =< X_1 X_2 ... X_n >$ is an ordered list of sets of items such that each set of items $X_i \subseteq L$. X_i is called a *sequence element.* All items in a sequence element are unordered. For short, we will also refer to a web access sequence as to a sequence.

We say that a web access sequence $< X_1 X_2 ... X_n >$ is *contained* in another web access sequence $< Y_1 Y_2 ... Y_m >$ if there exist integers $i_1 < i_2 < ... < i_n$ such that $X_1 \subseteq Y_{i_1}, X_2 \subseteq Y_{i_2}, ..., X_n \subseteq Y_{i_n}$.

Problem Formulation. Let D be a database of variable length web access sequences. Let S be a web access sequence. The problem of pattern queries consists in finding in D all web access sequences, which contain the web access sequence S.

1.2 Related Work

Database indexes provided today by most database systems are B^+-tree indexes to retrieve tuples of a relation with specified values involving one or more attributes [5]. Each non-leaf node contains entries of the form (v, p) where v is the separator value which is derived from the keys of the tuples and is used to tell which sub-tree holds the searched key, and p is the pointer to its child node. Each leaf node contains entries of the form (k, p), where p is the pointer to the tuple corresponding to the key k.

A set-based bitmap indexing, which is used to enable faster subset search in relational databases was presented in [14] (a special case of superimposed coding). The key idea of the set-based bitmap index is to build binary keys, called *group bitmap keys*, associated with each item set. The group bitmap key represents contents of the item set by setting bits to '1' on positions determined from item values (by means of *modulo* function). An example set-based bitmap index for three item sets: {0, 7, 12, 13}, {2, 4}, and {10, 15, 17} is given in Fig. 3. When a subset search query seeking for item sets containing e.g. items 15 and 17 is issued, the group bitmap key for the searched subset is computed (see Fig. 4). Then, by means of a bit-wise AND, the index is scanned for keys containing 1's on the same positions. As the result of the first step of the subset search procedure, the item sets identified by *set*=1 and *set*=3 are returned. Then, in the verification step (ambiguity of *modulo* function), these item sets are tested for the containment of the items 15 and 17. Finally, the item set identified by *set*=3 is the result of the subset search. Notice that this indexing method does not consider items ordering.

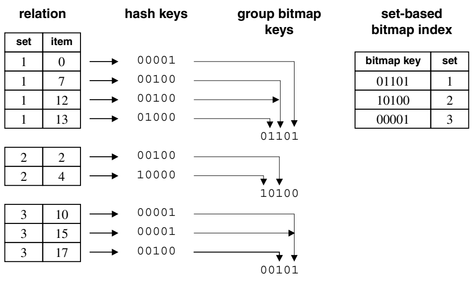


Fig. 3. Set-based bitmap index

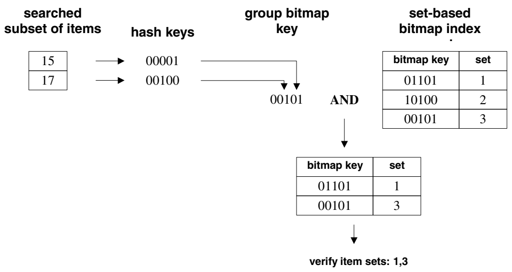


Fig. 4. Set retrieval using set-based bitmap index

In [8], a conceptual clustering method, using *entropic criterion for conceptual clustering* EC^3 is used to define indexing schemes on sets of binary features. Similar data item sets are stored in the same cluster, and similarity measure based on entropy is used during retrieval to find a cluster containing the searched subset. The method does not consider items ordering.

2 Sequential Index Structure

In this section we present our indexing method, called sequential indexing, for optimizing pattern queries. The sequential index structure consists of sequences of bitmaps generated for web access sequences. Each bitmap encodes all items (similarly to a set-based bitmap index) of a portion of a web access sequence as well as ordering relations between each two of the items.

We start with the preliminaries, then we present the index construction algorithm and explain how to use the sequential index structure. Finally, we discuss index storage and maintenance problems.

2.1 Preliminaries

Web access sequences contain categorical items in the form of URLs. For sake of convenience, we convert these items to integer values by means of an *item mapping function.*

Definition 1. *An item mapping function $fi(x)$, where x is a literal, is a function which transforms a literal into an integer value.*

Example 1. Given a set of literals $L = \{A, B, C, D, E, F\}$, an item mapping function can take the following values: $fi(A)$=1, $fi(B)$=2, $fi(C)$=3, $fi(D)$=4, $fi(E)$=5, $fi(F)$=6.

Similarly, we use an *order mapping function* to express web access sequence ordering relations by means of integer values. Thus, we will be able to represent web access sequence items as well as web access sequence ordering uniformly.

Definition 2. *An order mapping function $fo(x, y)$, where x and y are literals and $fo(x, y) \neq fo(y, x)$, is a function which transforms a web access sequence $< \{x\}\{y\} >$ into an integer value.*

Example 2. For the set of literals used in the previous example, an order mapping function can be expressed as: $fo(x, y) = 6 * fi(x) + fi(y)$, e.g. $fo(C, F) = 24$.

Using the above definitions, we will be able to transform web access sequences into item sets, which are easier to manage, search and index. An item set representing a web access sequence is called an *equivalent set.*

Definition 3. *An equivalent set E for a web access sequence $S =< X_1X_2... X_n >$ is defined as:*

$$E = \left(\bigcup_{x \in X_1 \cup X_2 \cup \ldots \cup X_n} \{fi(x)\} \right) \cup \left(\bigcup_{\substack{x,y \in X_1 \cup X_2 \cup \ldots \cup X_n: \\ x \ precedes \ y}} \{fo(x, y)\} \right) \qquad (1)$$

where: $fi()$ is an item mapping function and $fo()$ is an order mapping function.

Example 3. For the web access sequence $S =< \{A,B\}\{C\}\{D\} >$ and the presented item mapping function and order mapping function, the equivalent set E is evaluated as follows:

$$E = \left(\bigcup_{x\in\{A,B,C,D\}}\{fi(x)\}\right)\cup\left(\bigcup_{\substack{x,y\in\{<\{A\}\{C\}>,<\{B\}\{C\}>,\\<\{A\}\{D\}>,<\{B\}\{D\}>,<\{C\}\{D\}>\}}}\{fo(x,y)\}\right) =$$

$$= \{fi(A)\}\cup\{fi(B)\}\cup\{fi(C)\}\cup\{fi(D)\}\cup\{fo(A,C)\}\cup\{fo(B,C)\}\cup\{fo(A,D)\}\cup$$
$$\cup\,\{fo(B,D)\}\cup\{fo(C,D)\} = \{1,2,3,4,9,15,10,16,22\}$$

Observation. For any two web access sequences S_1 and S_2, we have: S_2 contains S_1 if $E_1 \subseteq E_2$, where E_1 is the equivalent set for S_1, and E_2 is the equivalent set for S_2. In general, this property is not reversible.

The size of the equivalent set depends on the number of items in the web access sequence and on the number of ordering relations between the items. For a given number of items in the web access sequence, the equivalent set will be the smallest if there are no ordering relations at all (i.e. $S =< X >$, then $|E| = |X|$, since $E = X$), and will be the largest if S is a sequence of one-item sets (i.e. $S =< X_1X_2...X_n >$, for all i we have $|X_i| = 1$, then $|E| = n + \binom{n}{2}$).

Since the size of an equivalent set quickly increases while increasing the number of the original sequence elements, we split web access sequences into *partitions*, which are small enough to process and encode.

Definition 4. *We say that a web access sequence* $S =< X_1X_2...X_n >$ *is* partitioned *into web access sequences* $S_1 =< X_1...X_{a_1} >$, $S_2 =< X_{a_1+1}...X_{a_2} >$, ..., $S_k =< X_{a_j+1}...X_n >$ *with level* β *if for each web access sequence* S_i *the size of its equivalent set* $|E_i| < \beta$ *and for all* $x, y \in X_1 \cup X_2 \cup ... \cup X_n$, *where* x *precedes* y, *we have: either* $< \{x\}\{y\} >$ *is contained in* S_i *or* $\{x\}$ *is contained in* S_i, *and* $\{y\}$ *is contained in* S_j, *where* $i < j$ *(*β *should be greater than maximal item set size).*

Example 4. Partitioning the web access sequence $S =< \{A,B\}\{C\}\{D\}\{A,F\}\{B\}\{E\} >$ with level 10 results in two web access sequences: $S_1 =< \{A,B\}\{C\}\{D\} >$ and $S_2 =< \{A,F\}\{B\}\{E\} >$, since the sizes of the equivalent sets are respectively: $|E_1| = 9$ ($E_1 = \{1,2,3,4,9,15,10,16,22\}$), and $|E_2| = 9$ ($E_2 = \{1,6,2,5,8,38,11,41,17\}$).

Observation. For a web access sequence S partitioned into S_1, S_2, ..., S_k, and a web access sequence Q, we have: S contains Q if there exists a partitioning of Q into Q_1, Q_2, ..., Q_m, such that Q_1 is contained in S_{i_1}, Q_2 is contained in S_{i_2}, ..., Q_m is contained in S_{i_m}, and $i_1 < i_2 < ... < i_m$.

Our sequential index structure will consist of equivalent sets stored for all web access sequences, optionally partitioned to reduce the complexity. To reduce storage requirements, equivalent sets will be stored in database in the form of *bitmap signatures*.

Definition 5. *The* bitmap signature *of a set X is an N-bit binary number created, by means of bit-wise OR operation, from the hash keys of all data items contained in X. The* hash key *of the item $x \in X$ is an N-bit binary number defined as follows: $hash\_key(X) = 2^{(X\ mod\ n)}$.*

Example 5. For the set $X = \{0, 7, 12, 13\}$, $N = 5$, the hash keys of the set items are the following:

$hash\_key(0) = 2^{(0\ mod\ 5)} = 1 = 00001$,
$hash\_key(7) = 2^{(7\ mod\ 5)} = 4 = 00100$,
$hash\_key(12) = 2^{(12\ mod\ 5)} = 4 = 00100$,
$hash\_key(13) = 2^{(13\ mod\ 5)} = 8 = 01000$.

The bitmap signature of the set X is the bit-wise OR of all items' hash keys:
$bitmap\_signature(X) = 00001\ OR\ 00100\ OR\ 00100\ OR\ 01000 = 01101$.

Observation. For any two sets X and Y, if $X \subseteq Y$ then:
$bitmap\_signature(X)\ AND\ bitmap\_signature(Y) = bitmap_signature(X)$,
where AND is a bit-wise AND operator. This property is not reversible in general (when we find that the above formula evaluates to $TRUE$ we still have to verify the result traditionally).

In order to plan the length N of a bitmap signature for a given average set size, consider the following analysis. Assuming uniform items distribution, the probability that representation of the set X sets k bits to '1' in an N-bit bitmap signature is:

$$P = \frac{\binom{N}{k} f_{k,|X|}}{N^{|X|}}, \text{ where } f_{0,|X|} = 0, f_{q,|X|} = q^{|X|} - \sum_{i=1}^{q-1} \binom{q}{i} f_{i,|X|} \qquad (2)$$

Example probabilistic expected value of number of bits set to '1' for a 16-bit bitmap signatures and various set sizes is illustrated in Fig. 5. We can observe that e.g. for a set of 10 items, N should be greater than 8 (else we have all bits set to 1 and the signature is unusable since it is always matched).

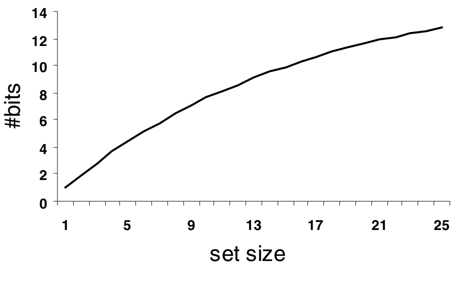


Fig. 5. Number of bitmap signature bits set to '1' for various set sizes (N=16)

The probability that a bitmap signature of the length N having k 1's matches another bitmap signature of the length N having m 1's is $\binom{m}{k}/\binom{N}{k}$. It means that the smaller k, the better pruning is performed during matching bitmap signatures of item sets, in order to check their containment (so we have to verify less item sets).

2.2 Sequential Index Construction Algorithm

The sequential index construction algorithm iteratively processes all web access sequences in the database. First, the web access sequences are partitioned with the given level β. Then, for each partition of each web access sequence, the equivalent set is evaluated. In the next step, for each equivalent set, its N-bit bitmap signature is generated and stored in the database. The formal description of the algorithm is given below.

Input: database D of web access sequences, partitioning level β, bitmap length N
Output: sequential index for D
Method:

```
for each web access sequence $S \in D$ do begin
    partition $S$ into partitions $S_1$, $S_2$, ..., $S_k$ with level $\beta$;
    for each partition $S_i$ do begin
        evaluate equivalent set $E_i$ for $S_i$;
        $bitmap_i = bitmap\_signature(E_i)$;
        store $bitmap_i$ in the database;
    end;
end.
```

Consider the following example of sequential index construction. Assume that β=10, N=16, and the database D contains three web access sequences: $S_1 = <\{A, B\}\{C\}\{D\}\{A, F\}\{B\}\{E\}>$, $S_2 =<\{A\}\{C, E\}\{F\}\{B\}\{E\}\{A, D\}>$, $S_3 =<\{B, C, D\}, \{A\}>$.

First, we partition the web access sequences with β=10. Notice that S_3 is, in fact, not partitioned since its equivalent set is small enough. The symbol $S_{i,j}$ denotes j-th partition of the i-th web access sequence.

$S_{1,1} =<\{A, B\}\{C\}\{D\}>$ (ordering relations are: $A \to C$, $B \to C$, $A \to D$, $B \to D$, $C \to D$)
$S_{1,2} =<\{A, F\}\{B\}\{E\}>$ (ordering relations are: $A \to B$, $F \to B$, $A \to E$, $F \to E$, $B \to E$)
$S_{2,1} =<\{A\}\{C, E\}\{F\}>$ (ordering relations are: $A \to E$, $A \to C$, $E \to F$, $C \to F$)
$S_{2,2} =<\{B\}\{E\}\{A, D\}>$ (ordering relations are: $B \to E$, $B \to A$, $B \to D$, $E \to A$, $E \to D$)
$S_{3,1} =<\{B, C, D\}\{A\}>$ (ordering relations are: $B \to A$, $C \to A$, $D \to A$)

Then we evaluate the equivalent sets for the partitioned web access sequences. We use the example item mapping function and order mapping function taken from the Definitions 1 and 2. The symbol $E_{i,j}$ denotes the equivalent set for $S_{i,j}$.

$E_{1,1} = \{1, 2, 3, 4, 9, 15, 10, 16, 22\}$
$E_{1,2} = \{1, 6, 2, 5, 8, 38, 11, 41, 17\}$
$E_{2,1} = \{1, 3, 5, 6, 11, 9, 36, 24\}$

$E_{2,2} = \{2, 5, 1, 4, 17, 13, 16, 31, 36\}$

$E_{3,1} = \{2, 3, 4, 1, 13, 19, 25\}$

In the next step, we generate 16-bit bitmap signatures for all equivalent sets.

$bitmap\_signature(E_{1,1}) = 1000011001011111$

$bitmap\_signature(E_{1,2}) = 0000101101100110$

$bitmap\_signature(E_{2,1}) = 0000101101111010$

$bitmap\_signature(E_{2,2}) = 1010000000110111$

$bitmap\_signature(E_{3,1}) = 0010001000011110$

Finally, the sequential index is stored in the database in the following form:

SID	bitmap_signature
1	1000011001011111, 0000101101100110
2	0000101101111010, 1010000000110111
3	0010001000011110

2.3 Using Sequential Index for Pattern Queries

During pattern query execution, the bitmap signatures for all web access sequences are scanned. For each web access sequence, the test of a searched subsequence mapping is performed. If the searched subsequence can be successfully mapped to the web access sequence partitions, then the web access sequence is read from the database. Due to the ambiguity of bitmap signature representation, additional verification of the retrieved web access sequence is required. The verification can be performed using the traditional B^+-tree method, since it consists in reading the web access sequence from the database and checking whether it contains the searched subsequence. The formal description of the algorithm is given below. We use a simplified notation of $Q[i\_start..i\_end]$ to denote a partition $< X_{i\_start} X_{i\_start+1} ... X_{i\_end} >$ of a sequence $Q =< X_1 X_2 ... X_n >$, where $1 \leq i\_start \leq i\_end \leq n$. The symbol & denotes bit-wise AND operation.

```
Input: sequential index, searched subsequence $Q$
Output: identifiers of web access sequences to be verified
Method:
  for each sequence identifier $sid$ do begin
    $j = 1$;
    $i\_end = 1$;
    repeat
      $i\_start = i\_end$;
      evaluate equivalence set $E_Q$ for $Q[i\_start..i\_end]$;
      $mask$ = bitmap_signature($E_Q$);
      while $mask$ & bitmap_signature($E_{sid,i}$) <> $mask$
              and $j \leq$ number of partitions for $sid$ do $j$++;
      if $j \leq$ number of partitions for $sid$ then repeat
        $i\_end$++;
        generate equivalence set $E_Q$ for $Q[i\_start..i\_end]$;
```

```
            $mask$ = bitmap_signature($E_Q$);
         until $mask$ & bitmap_signature($E_{sid,i}$) <> $mask$
            or $i\_end$ = size of $Q$;
      until $i\_start$ = $i\_end$ or $j >$ number of partitions for $sid$;
      if $j \leq$ number of partitions then return($sid$);
   end.
```

Consider the following example of using sequential index to perform pattern queries. Assume that we look for all web access sequences, which contain the subsequence $<\{F\}\{B\}\{D\}>$. We begin with *sid*=1. We find that $<\{F\}>$ (0000001000000000) matches the first partition (1111101001100001). So, we check whether $<\{F\},\{B\}>$ (0010001000000000) also matches this partition. Accidentally it does, but when we try $<\{F\},\{B\},\{D\}>$ (1010101010000000), we find that it does not match the first partition. Then we move to the second partition to check whether $<\{D\}>$ (0000100000000000) matches the partition (0110011011010000). This test fails and since we have no more partitions, we reject *sid*=1 (this web access sequence does not contain the given subsequence).

In the next step, we check *sid*=2. We find that $<\{F\}>$ (0000001000000000) matches the first partition (0101111011010000). So, we check whether $<\{F\},\{B\}>$ (0010001000000000) also matches this partition. It does not, so we move to the second partition and find that $<\{B\}>$ (0010000000000000) matches the partition (1110110000000101). Then we must check whether $<\{B\},\{D\}>$ (1010100000000000) also matches the partition. This time the check is positive and since we have matched the whole subsequence, we return *sid*=2 as a part of the result. The web access sequence will be verified later.

Finally, we check *sid*=3. We find that $<\{F\}>$ (0000001000000000) does not match the first partition (0111100001000100). Since we have no more partitions, we reject *sid*=3 (this web access sequence does not contain the given subsequence).

So far, the result of our index scanning is the web access sequence identified by *sid*=2. We still need to read and verify, whether the sequence really contains the searched subset. In our example it does, so the result is returned to the user.

2.4 Physical Storage

Since a sequential index is fully scanned each time a pattern query is performed, it is critical to store it efficiently. We store index entries in the form of $< p, n, bitmap_1, bitmap_2, ..., bitmap_n >$, where p is a pointer to a web access sequence described by the index entry, n is the number of bitmap signatures, and $bitmap_i$ is a single bitmap signature for the web access sequence. The pointer p should address the translation table, which contains pointers to physical tuples of the relation holding the web access sequences (the structure is $< n, p_1, p_2, .., p_n >$). Since we usually have a B^+-tree index on a sequence identifier attribute (to optimize joins), we can use its leaves can as a translation table instead of consuming database space by redundant structures. Example storage implementation for the sequential index from Sect. 2.2 is given in Fig. 6.

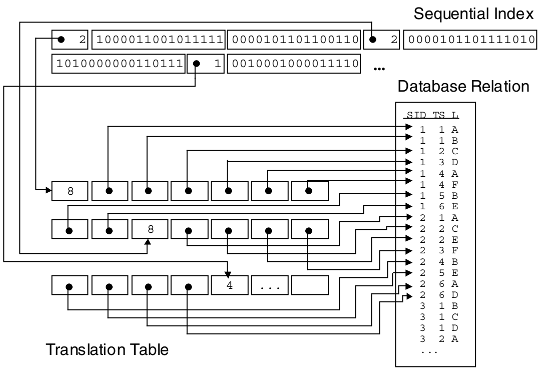


Fig. 6. Example physical storage structure for sequential index

2.5 Update Operations

Maintenance of a sequential index is quite expensive, since bitmap signatures are not reversible, and updates may influence partitioning of web access sequences. For example, when we insert a new tuple into the database, thus extending a web access sequence, we cannot determine what partition should the tuple belong to. Similarly, when we delete a tuple, then both we cannot determine the corresponding partition, and, even if we could do it, we do not know, whether the item being deleted was the only item mapped to a given bit of the bitmap signature (so we could reset the bit).

In order to have a consistent state of a sequential index, we must perform the complete index creation procedure (partitioning, evaluating equivalent sets, generating bitmap signatures) for the web access sequence being modified. However, since this solution might reduce DBMS performance for transaction-intensive databases, we propose the following algorithm of *offline maintenance* for sequential indexes:

1. Whenever a new item is added to an existing web access sequence, we set to '1' all bits in the first bitmap signature for the web access sequence. It means that any subsequence will match the first bitmap signature, and therefore we will not miss the right one. Any false hits will be eliminated during actual verification of subsequence containment.
2. Whenever an item is removed from an existing web access sequence, we do not perform any modifications on the bitmap signatures of the web access sequence. We may get false hits, but they will be eliminated during final verification.

Notice that using the above algorithm, the overall index performance may decrease temporarily, but we will not get incorrect query results. Over a period of time, the index should be rebuild either completely, or for updated web access sequences only, e.g. according to a transaction log.

3 Experimental Results

We have performed several experiments on synthetic data sets to evaluate our sequential indexing method. The database of web access sequences was generated randomly, with uniform item distribution, and stored by *Oracle8* DBMS. We used dense data sets, i.e. the number of available items was relatively small, and therefore each item occurred in a large number of web access sequences. The web access sequences contained 1-item sets only (pessimistic approach - maximal number of ordering relations).

Figure 7a shows the number of disk blocks (including index scanning and relation access), which were read in order to retrieve web access sequences containing subsequences of various lengths. The data set contained 50000 web access sequences, having 20 items of 50 in average. The compared database accessing methods were: traditional *SQL* query using B^+-tree index on *IP* attribute (B^+-tree), 24-bit set-based bitmap index (24S), 32-bit sequential index with $\beta = 28$ built on top of 24-bit set-based bitmap index (24S32Q28), and 48-bit sequential index with $\beta = 55$ built on top of 24-bit set-based bitmap index (24S48Q55). Our sequential index achieved a significant improvement for the searched subsequences of length greater than 4, e.g. for the subsequence length of 5 we were over 20 times faster than the B^+-tree method and 8 times faster than the set-based bitmap index.

We also analyzed the influence of the partitioning level β value on the sequential index performance. Figure 7b illustrates the filtering factor (percentage of web access sequences matched) for three sequential indexes built on bitmap signatures of total size of 48 bits, but with different partitioning. We noticed that partitioning web access sequences into a large number of partitions (small β) results in performance increase for long subsequences, but worsens the performance for short subsequences. Using a small number of web access sequence partitions (high β) results in more "stable" performance, but the performance is worse for long subsequences.

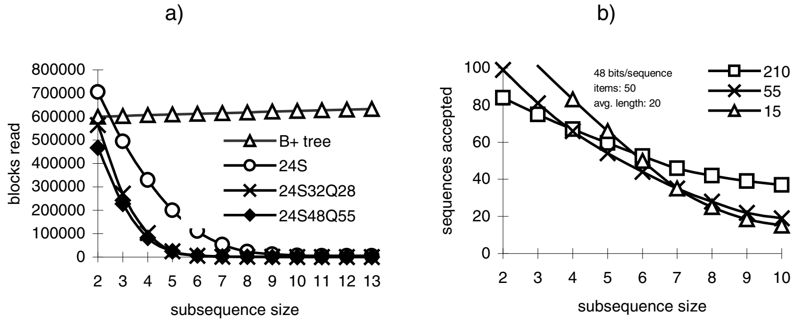


Fig. 7. Experimental results

4 Final Conclusions

Pattern queries on web access logs are specific in the sense that they require complicated *SQL* queries and database access methods (multiple joins, inefficient optimization). In this paper we have presented the new indexing method, called sequential indexing, which can replace a B^+-tree indexing and set-based indexing. During experiments, we have found that the most efficient solution is to combine a set-based index (which checks items of a web access sequence) with a sequential index (which checks the items ordering), what results in dramatic outperforming B^+-tree access methods.

References

1. Agrawal R., Srikant R.: Mining Sequential Patterns. Proc. of the 11th ICDE Conf. (1995)
2. Bayardo R.J.: Efficiently Mining Long Patterns from Databases. Proc. of the ACM SIGMOD International Conf. on Management of Data (1998)
3. Bentley J.L.: Multidimensional binary search trees used for associative searching. Comm. of the ACM 18 (1975)
4. Catledge L.D., Pitkow J.E.: Characterizing Browsing Strategies in the World Wide Web. Proc. of the 3rd Int'l WWW Conference (1995)
5. Comer D.: The Ubiquitous B-tree. Comput. Surv. 11 (1979)
6. Cooley R., Mobasher B., Srivastava J.: Data preparation for mining World Wide Web browsing patterns. Journal of Knowledge and Information Systems 1 (1999)
7. Cooley R., Mobasher B., Srivastava J.: Grouping Web Page References into Transactions for Mining World Wide Web Browsing Patterns. Proc. of the 1997 IEEE Knowledge and Data Engineering Exchange Workshop (1997)
8. Diamantini C., Panti M.: A Conceptual Indexing Method for Content-Based Retrieval. Proc. of the 15th IEEE Int'l Conf. on Data Engineering (1999)
9. Guralnik V., Wijesekera D., Srivastava J.: Pattern Directed Mining of Sequence Data. Proc. of the 4th KDD Conference (1998)
10. Guttman A.: R-trees: A dynamic index structure for spatial searching. Proc. of ACM SIGMOD International Conf. on Management of Data (1984)
11. Luotonen A.: The common log file format. `http://www.w3.org/pub/WWW/` (1995)
12. Mannila H., Toivonen H.: Discovering generalized episodes using minimal occurrences. Proc. of the 2nd KDD Conference (1996)
13. Mannila H., Toivonen H., Verkamo A.I.: Discovering frequent episodes in sequences. Proc. of the 1st KDD Conference (1995)
14. Morzy T., Zakrzewicz M.: Group Bitmap Index: A Structure for Association Rules Retrieval. Proc. of the 4th KDD Conference (1998)
15. O'Neil P.: Model 204 Architecture and Performance. Proc. of the 2nd International Workshop on High Performance Transactions Systems (1987)
16. Perkowitz M., Etzioni O.: Adaptive Web Sites: an AI challenge. Proc. of the 15th Int. Joint Conf. AI (1997)
17. Pirolli P., Pitkow J., Rao R.: Silk From a Sow's Ear: Extracting Usable Structure from the World Wide Web. Proc. of Conf. on Human Factors in Computing Systems (1996)
18. Pitkow J.: In search of reliable usage data on the www. Proc. of the 6th Int'l WWW Conference (1997)

19. Srikant R., Agrawal R.: Mining Sequential Patterns: Generalizations and Performance Improvements. Proc. of the 5th EDBT Conference (1996)
20. Yan T.W., Jacobsen M., Garcia-Molina H., Dayal U.: From User Access Patterns to Dynamic Hypertext Linking. Proc. of the 5th Int'l WWW Conference (1996)

Ensemble Feature Selection Based on Contextual Merit and Correlation Heuristics[1]

Seppo Puuronen, Iryna Skrypnyk, and Alexey Tsymbal

Department of Computer Science and Information Systems, University of Jyväskylä
P.O. Box 35, FIN-40351 Jyväskylä, Finland
{sepi,iryna,alexey}@cs.jyu.fi

Abstract. Recent research has proven the benefits of using ensembles of classifiers for classification problems. Ensembles of diverse and accurate base classifiers are constructed by machine learning methods manipulating the training sets. One way to manipulate the training set is to use feature selection heuristics generating the base classifiers. In this paper we examine two of them: correlation-based and contextual merit -based heuristics. Both rely on quite similar assumptions concerning heterogeneous classification problems. Experiments are considered on several data sets from UCI Repository. We construct fixed number of base classifiers over selected feature subsets and refine the ensemble iteratively promoting diversity of the base classifiers and relying on global accuracy growth. According to the experimental results, contextual merit -based ensemble outperforms correlation-based ensemble as well as C4.5. Correlation-based ensemble produces more diverse and simple base classifiers, and the iterations promoting diversity have not so evident effect as for contextual merit -based ensemble.

1 Introduction

Machine learning research has progressed in many directions. One of those directions still containing a number of open questions is the construction and use of an ensemble of classifiers. Ensembles are well established as a method for obtaining highly accurate combined classifiers by integrating less accurate base classifiers.

Many methods for constructing ensembles have been developed. They can be divided into two main types: general methods and methods specific to a particular learning algorithm. Amongst successful general ones are sampling methods, and methods manipulating either the input features or the output targets. We apply the second type of method when ensemble integration is made using weighted voting.

The goal of traditional feature selection is to find and remove features that are unhelpful or destructive to learning in order to construct a *single* classifier [4]. Since an

[1] This research is partially supported by the Academy of Finland (project #52599), the Centre for International Mobility (CIMO) and the COMAS Graduate School of the University of Jyväskylä.

ensemble of classifiers represents multiple inference models, it can produce different inference models in different sub areas of the instance space. Feature selection heuristics are used to assist in guiding the process of constructing base classifiers implementing those models. In addition to decreasing the number of features, ensemble feature selection has an additional goal of finding *multiple* feature subsets to produce a *set* of base classifiers promoting disagreement among them [15].

This paper continues our research in ensemble feature selection [18,20]. We have developed an algorithm with an iterative refinement cycle. This cycle provides feedback to ensemble construction in order to improve ensemble characteristics with selected feature selection heuristics. In this paper we examine two feature selection heuristics for ensemble creation applying also the refinement cycle.

For our study we chose two heuristics for feature selection that have an ability to treat complex interrelations between features, namely contextual merit-based heuristic and correlation-based heuristic [1,7,9,16]. Interrelations between features that are assumed in those heuristic might be a consequence of the fact, that some domains contain features varying in importance across the instance space [1,7,10,12,16]. This situation, called *feature-space heterogeneity*, is wide spread for real data sets. Heterogeneity in data becomes critical, because most classification algorithms fail to make accurate predictions.

We compare correlation-based and contextual merit -based heuristic on several data sets from UCI repository and make some conclusions about the use of those heuristics for ensemble feature selection. Particularly, we observe co-ordination of feature selection heuristics and internal design of the refinement cycle.

In Chapter 2 we consider the basic framework of ensemble classification. Chapter 3 describes the Contextual Merit measure (CM measure) and correlation-based merit measure. Chapter 4 presents brief description of the algorithm for ensemble feature selection with the iterative refinement cycle. In Chapter 5 our experimental study on several data sets is described. We summarize with conclusions in Chapter 6.

2 Ensemble Classification

In supervised learning, a learning algorithm is given training instances of the form $\{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_M, y_M)\}$ for some unknown function $y = g(\mathbf{x})$, where $\mathbf{x}_i$ values are vectors of the form $\langle \mathrm{x}_{i,1}, \ldots, \mathrm{x}_{i,j}, \ldots, \mathrm{x}_{i,N} \rangle$, where $x_{i,j}$ are feature values of $\mathbf{x}_i$, and M is the size of the training set $\mathbf{T}$. In this paper we will refer to features as f_j, $j = 1 \ldots N$, where N is the number of features. Given a set of training instances $\mathbf{T}$, a learning algorithm outputs a *classifier h*, which is a hypothesis about the true function g. Given new $\mathbf{x}$ values, it predicts the corresponding y values. In ensemble classification (Fig. 1) a set of base classifiers $h_1, \ldots, h_S$, where S is the size of the ensemble is formed during *the learning phase*.

Each base classifier in the ensemble (classifiers $h_1 \ldots h_S$ in this case) is trained using training instances of the corresponding training set $\mathbf{T}i$, $i = 1, \ldots, S$. In this paper we form the training sets $\mathbf{T}_1, \ldots, \mathbf{T}_S$ as the subsets of features using feature selection heuristics. For the ensemble classification the classifications of the base classifiers are combined during *the application phase* in some way $h^* = F(h_1, h_2, \ldots, h_S)$ to produce the final classification of the ensemble. In this paper the final classification y^* is formed using weighted voting [2,3,5,8].

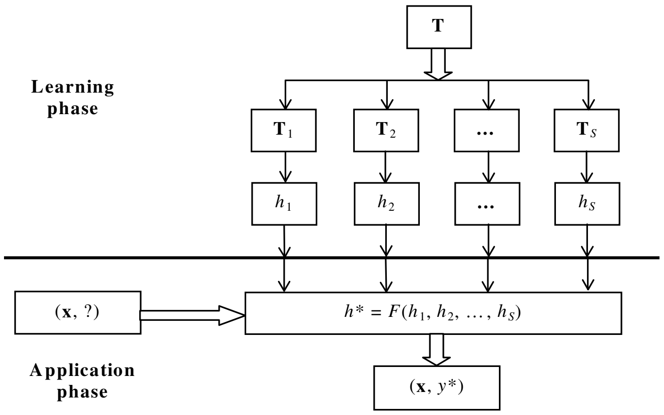


Fig. 1. Ensemble classification process

Research has shown that an effective ensemble should consist of a set of base classifiers that not only have high accuracy, but also make their errors on different parts of the input space as well [5,14].

Combining the classifications of several classifiers is useful only if there is disagreement among the base classifiers, i.e. they are independent in the production of their errors. The error rate of each base classifier should not exceed a certain limit. Otherwise, the ensemble error rate will usually increase as a result of combination of their classifications [5]. The measure of independence of the base classifiers is called the *diversity* of an ensemble. Several ways to calculate the diversity are considered in [6,15,17].

3 Heuristics for Ensemble Feature Selection

In this chapter we consider two heuristics for feature selection, namely the contextual merit measure and correlation-based merit measure which are intended to treat feature-space heterogeneity in classification problems [1]. Both of them are independent on the learning algorithm.

3.1 Contextual Merit Measure

The main assumption of CM measure, which was developed in [9] is that features important for classification should differ significantly in their values to predict instances from different classes. The CM measure is robust to both problems of class heterogeneity and feature-space heterogeneity. The CM measure assigns a merit value to a feature taking into account the degree to which the other features are capable to discriminate between the same instances as the given feature. In an extreme situation, when two instances of different classes differ in only one feature than that feature is particularly valuable for classification and it is assigned additional merit.

We use the CM measure as it has been presented in [9] and described below. Let the difference $d_{\mathbf{x}_i,\mathbf{x}_k}^{(f_j)}$ between the values of a categorical feature f_j for the vectors $\mathbf{x}_i$ and $\mathbf{x}_k$ be

$$d_{\mathbf{x}_i,\mathbf{x}_k}^{(f_j)} = \begin{array}{l} 0, \text{ if the values are same} \\ 1, \text{ otherwise.} \end{array} \tag{1}$$

and between the values of a numeric feature f_j correspondingly be

$$d_{\mathrm{x}_i,\mathrm{x}_k}^{(f_j)} = \min\left(\left| \mathrm{x}_{i,j} - \mathrm{x}_{k,j} \right| \big/ t_{f_j},\ 1 \right) \tag{2}$$

where $\mathrm{x}_{i,j}$ is the value of the feature f_j in the vector $\mathbf{x}_i$, and t_{f_j} is a threshold. In this paper it is selected to be one-half of the range of the values of the feature f_j.

Then the distance $D_{\mathbf{x}_i,\mathbf{x}_k}$ between the vectors $\mathbf{x}_i$ and $\mathbf{x}_k$ is

$$D_{\mathbf{x}_i,\mathbf{x}_k} = \sum_{j=1}^{N} d_{\mathbf{x}_i,\mathbf{x}_k}^{(f_j)} \tag{3}$$

where N is the number of features. The value of CM measure CM_{f_j} of a feature f_j is

$$CM_{f_j} = \sum_{i=1}^{M} \sum_{\mathbf{x}_k \in \overline{C(\mathbf{x}_i)}} w_{\mathbf{x}_i,\mathbf{x}_k}^{(f_j)} d_{\mathbf{x}_i,\mathbf{x}_k}^{(f_j)} \tag{4}$$

where M is the number of instances, $\overline{C(\mathbf{x}_i)}$ is the set of vectors not from the same class as the vector $\mathbf{x}_i$, and $w_{\mathbf{x}_i,\mathbf{x}_k}^{(f_j)}$ is a weight chosen so that instances that are close to each other, i.e., that differ only in a few of their features, have greater influence in determining each feature's CM measure value. In [9] weights $w_{\mathbf{x}_i,\mathbf{x}_k}^{(f_j)} = 1/D_{\mathbf{x}_i,\mathbf{x}_k}^{2}$ were used when $\mathbf{x}_k$ is one of the K nearest neighbors of $\mathbf{x}_i$, in terms of $D_{\mathbf{x}_i,\mathbf{x}_k}$, in the set $\overline{C(\mathbf{x}_i)}$, and $w_{\mathbf{x}_i,\mathbf{x}_k}^{(f_j)} = 0$ otherwise. The number of nearest neighbors k used in [9] was the binary logarithm of the number of examples in the set $\overline{C(\mathbf{x}_i)}$.

3.2 Correlation-Based Merit Measure

Correlation-based approach is the other widespread approach to estimate interrelations between features, or features and the class variable. It uses the Pearson's correlation coefficient as a measure of linear dependence between two variables. Such kind of dependence is quite common in real world situations.

Many researchers have used the correlation-based approach to estimate the goodness of feature subsets. For example, [16,21] used correlation between particular feature and particular class, and features with highest correlation were selected to construct a classifier for that class separately producing different models for different classes. We will estimate the goodness of feature subset as it was done in [7]. The correlation -based merit measure will be calculated not for a particular feature, but for a particular feature subset. In that way we will be able to consider interrelations between features in the subset, too. The basic assumption for this heuristic is as follows. Good feature subsets contain features highly correlated with the class, yet uncorrelated with each other. The correlation coefficient $r_{f_j f_k}$ between two numerical features f_j and f_k is calculated using the formula (5).

$$r = \frac{S_{f_j f_k}}{S_{f_j} S_{f_k}} \tag{5}$$

where $S_{f_j f_k}$ is a sample covariance between features f_j and f_k, and S_{f_j} and S_{f_k} are sample standard deviations for features f_j and f_k correspondingly.

According to [7] we calculate the merit of a feature subset M_F with the formula (6).

$$M_F = \frac{n \left|\bar{r}_{f_j y}\right|}{\sqrt{n + n(n-1)\left|\bar{r}_{f_j f_k}\right|}} \tag{6}$$

where F is a feature subset containing n features, y is a class variable, $\bar{r}_{f_j y}$ is the mean feature-class correlation ($f_j \in F$), and $\bar{r}_{f_j f_k}$ is the average feature-feature intercorrelation. The numerator in formula (6) gives an indication of how predictive for the class the subset of features is, and the denominator expresses how much redundancy there exist. Formula (5) is suitable only for the case of numeric variables. For categorical values binarization is used.

Let f_j be a categorical feature having t values $\text{v}_1, \ldots, \text{v}_t$. We form t binary attributes f_j^i, $i = 1, \ldots, t$ so that $f_j^i = 1$, when $f_j = \text{v}_i$ and $f_j^i = 0$ otherwise. The correlation between the categorical feature f_j and the numeric feature f_k is then calculated using the formula (7).

$$r_{f_j f_k} = \overset{t}{\underset{i}{}} \; p\left(f_j = \text{v}_j\right) \, r_{f_j^i f_k} \tag{7}$$

Let f_j be a categorical feature having t values $\text{v}_1, \ldots, \text{v}_t$ and f_k be a categorical feature having l values $\text{u}_1, \ldots, \text{u}_l$. The binary attributes f_j^i, $i = 1, \ldots, t$ and f_k^q, $q = 1, \ldots, l$ are formed as described above. The correlation between these two categorical features is then calculated using the formula (8).

$$r_{f_j f_k} = \overset{t}{\underset{i}{}} \quad \overset{l}{\underset{q}{}} \; p\left(f_j = \mathrm{v}_i , f_k = \mathrm{u}_q\right) \; r_{f_j^i f_k^q} \tag{8}$$

In addition, the two formulae above are robust to missing values [7].

4 An Algorithm for Ensemble Feature Selection

In this chapter we present an overview of our algorithm *EFS_ref* for ensemble feature selection with the iterative refinement cycle. An algorithm for ensemble feature selection with contextual merit-based heuristic was considered in detail also in [18]. In this paper we extend it to the correlation-based heuristic, too. For the two heuristics the algorithm constructs an ensemble of a fixed number of base classifiers over selected feature subsets. The number of base classifiers in both ensembles is the number of different classes among the instances of the training set. The objective is to build each base classifier on the feature subset including features most relevant for distinguishing the corresponding class from the others. It is necessary to note that in this algorithm the base classifiers are still not binary, but distinguish the whole number of classes present in the data set. Each base classifier of the initial ensemble is based on a fixed number of features with the highest value of the CM measure for the corresponding class, or with the highest correlation-based measure correspondingly. The initial ensemble is iteratively modified trying to change the number of features one by one for the less diverse base classifiers suggesting exclusions or inclusions of features. With the correlation-based measure candidate feature subsets for new base classifiers are formed using forward inclusion procedure. The iterations are guided by the diversity of classifiers and the value of the CM measure, or correlation-based merit measure depending on the heuristic used.

For our algorithm we use the diversity measure calculated as an average difference in predictions between all pairs of classifiers. The diversity is calculated similarly as in [17] where it was used as a measure of independence of the base classifiers. The modified formula to calculate the approximated diversity Div_i of a classifier h_i is (9).

$$Div_i = \frac{\overset{M}{\underset{j=1}{}} \quad \overset{S}{\underset{k=1, k\neq i}{}} Dif\left(h_i\left(\mathbf{x}_j\right), h_k\left(\mathbf{x}_j\right)\right)}{M \; \left(S-1\right)} \tag{9}$$

where S denotes the number of the base classifiers, $h_i(\mathbf{x}_j)$ denotes the classification of the vector $\mathbf{x}_j$ by the classifier h_i, and $Dif(a,b)$ is zero if the classifications a and b are same and one if they are different, and M is the number of instances in the test set. The diversity of an ensemble is calculated as the average diversity of all the base classifiers (10).

$$Div = \frac{\overset{S}{\underset{i=1}{}} Div_i}{S} \tag{10}$$

The CM measure is calculated as in [9] and described in Sect. 3.1. The correlation measure is calculated as described in Sect. 3.2.

Our algorithm is composed of two main phases: 1) the construction of the initial ensemble and 2) the iterative development of new candidate ensembles. Let DS be a data set including instances $\{(\mathbf{x}_1, y_1),\ldots, (\mathbf{x}_M, y_M)\}$, where $\mathbf{x}_i = \langle x_{i,1}, \ldots, x_{i,N}\rangle$ is the vector including the values of each of the N features. The main outline of our algorithm is presented below.

```
Algorithm EFS_ref(DS)

DS        the whole data set
TRS       training set
VS        validation set used during the iterations
TS        test set
Ccurr     the current ensemble of base classifiers
Accu      the accuracy of the final ensemble
FS        set of feature subsets for base classifiers
Threshold Threshold value used to select the features for the
          initial ensemble
begin
  divide_instances(DS,TRS,VS,TS) {divides DS into TRS,
  VS, and TS using stratified random sampling}
  for Heuristic∈{CM,Corr}
     Ccurr=build_initial_ensemble(Heuristic,TRS,FS,
     Threshold)
     loop
        cycle(Heuristic,TRS,Ccurr,FS,VS)
        {developes candidate ensembles and updates
        Accu,Ccurr,and FS when necessary}
     until no_changes
     Accu=accuracy(TS,Ccurr)
  end for
end algorithm EFS_ref
```

In the *EFS_ref* algorithm ensembles are generated in two places, the initial ensembles are generated using a different procedure than the generation of the new candidate ensembles that is included in the procedure *cycle*. One of the two initial ensembles is constructed using CM-based merit values and the other one using correlation-based merit values. In all cases the features with the highest normalized merit values up to the threshold are selected and the base classifiers are built using the C4.5 learning algorithm. The threshold is fixed ($\in\{0.1,0.2,\ldots,0.9\}$) in advance so that the accuracy of the ensemble over the training set is the highest one with the weighted voting. The main outline of the initial ensemble algorithm is presented below. The threshold value is used to cut the interval of merit values in order to select the features. For example the threshold value 0.1 means that only those features whose merit values are in the interval of the highest 10% of the whole interval of the merit values are selected.

```
build_initial_ensemble(Heuristic,TRS,FS,Threshold)

  L     number of classes and number of base classifiers
   begin
     Ensemble=∅;FS=∅
```

```
      for i from 1 to L
         MERITS[i]=calculate_merits(Heuristic,TRS,i)
         {feature merits for class i}
         FS[i]=select_features(MERITS,Threshold){selects
         features with the highest merits using threshold}
         C[i]=C4.5(TRS,FS[i]){learns classifier i}
         Ensemble=Ensemble∪C[i];FS=FS∪FS[i]
      end for
   end build_initial_ensemble=Ensemble
```

The iteration mechanism used in this paper changes one base classifier in each iteration. The base classifier with the smallest diversity is taken as the potential classifier to be changed. For the potential classifier, one feature is tried to be added or deleted from the subset of features that was used to train the classifier. The feature is selected using the merit value. The outline of the iteration algorithm is described below.

```
cycle(Heuristic,TRS,Ccurr,FS,VS)
begin
   loop
      for i from 1 to L
         DIV[i]=calculate_diversities(TRS,Ccurr)
      end for
         Cmin= argmin DIV[i]
         NewCcurr=Ccurr\C[Cmin];NewFS=FS
         MERITS[Cmin]=calculate_merits(Heuristic,TRS,Cmin)
         NewFS[Cmin]=FS[Cmin]\feature with min MERITS[Cmin]
               included in FS[Cmin]
         NewCcurr=NewCcurr∪C4.5(TRS,NewFS[Cmin])
         if accuracy  (VS,NewCcurr)>=accuracy(VS,Ccurr)
            then Ccurr=NewCcurr; FS=NewFS
            else no changes to Ccurr and FS
   until no change
   loop
      for i from 1 to L
         DIV=calculate_diversities(TRS,Ccurr)
      end for
         Cmin= argmin DIV[i]
         NewCcurr=Ccurr\C[Cmin];NewFS=FS
         MERITS[Cmin]=calculate_merits(Heuristic,TRS,Cmin)
         NewFS[Cmin]=FS[Cmin]∪feature with max MERITS[Cmin]
               not included in FS[Cmin]
         NewCcurr=NewCcurr∪C4.5(TRS,NewFS[Cmin])
         if accuracy  (VS,NewCcurr)>=accuracy(VS,Ccurr)
            then Ccurr=NewCcurr; FS=NewFS
            else no changes to Ccurr and FS
   until no change
end cycle
```

The iteration mechanism is composed of two loops. Both loops try to replace one base classifier at a time. The base classifier, which is tried to be replaced is the one with the lowest diversity value. When the classifier to be changed has been selected, one feature is either added (second loop) or deleted (first loop) from the subset of features that was used to train the classifier. The feature suggested to be added or deleted is selected using merit value of the corresponding heuristic. The accuracy of the previous ensemble is compared with the accuracy of the changed ensemble and the change is accepted if the first one is not higher. Both loops end when there were no changes accepted during the whole cycle.

5 Experiments

In this chapter, experiments with our algorithm for generation of an ensemble of classifiers built on different feature subsets are presented. First, the experimental setting is described, and then, results of the experiments are presented. The experiments are conducted on ten data sets taken from the UCI machine learning repository [13]. These data sets were chosen so as to provide a variety of application areas, sizes, combinations of feature types, and difficulty as measured by the accuracy achieved on them by current algorithms.

For each data set 30 test runs are made. In each run the data set is first split into the training set and two test sets by stratified random sampling keeping the class distribution of instances in each set approximately the same as in the initial data set. The training set (TRS) includes 60 percent of instances and the test sets (VS and TS) both 20 percent of instances. The first test set, VS (validation set) is used for tuning the ensemble of classifiers, adjusting the initial feature subsets so that the ensemble accuracy becomes as high as possible using the selected heuristic. The other test set, TS is used for the final estimation of the ensemble accuracy. The base classifiers themselves are learnt using the C4.5 decision tree algorithm with pruning [19] and the test environment was implemented within the MLC++ framework [11].

Our aim is to analyze the contribution of the iterative refinement cycle for the CM- and correlation-based ensembles, and then compare their accuracy with C4.5 accuracy. We also examine which of two feature selection heuristics produces more diverse and more accurate ensembles.

Table 1. Accuracies (%) of the CM-based and correlation-based ensembles, and C4.5

Data set	CM -based			Correlation -based			C4.5
	before	after	threshold	before	after	threshold	
Car	86.6	87.9	0.5	86.0	86.8	0.6	87.9
Glass	64.8	65.2	0.1	65.7	66.0	0.3	62.6
Iris	94.6	94.3	0.9	94.1	94.2	0.7	93.9
LED_17	64.4	64.6	0.1	59.5	62.1	0.4	65.0
Lymph	76.4	75.3	0.5	74.0	73.6	0.6	74.2
Thyroid	91.4	92.7	0.6	93.4	93.1	0.3	92.8
Wine	93.5	93.5	0.2	92.3	93.4	0.7	92.9
Waveform	73.0	74.2	0.2	74.3	74.2	0.5	72.2
Vehicle	67.0	67.3	0.1	67.6	68.1	0.5	68.7
Zoo	92.2	93.5	0.1	87.0	88.1	0.5	92.4

We collected accuracies (Table 1) and diversities (Table 2) for these two ensembles on each data set before and after iterations marking the corresponding columns as CM_b and CM_a for CM-based ensemble, and CR_b and CR_a for correlation-based ensemble. Thresholds used for each data set are presented in the corresponding columns for CM-based and correlation-based ensembles.

Table 2. Diversities of the base classifiers before and after iterations for the CM-based and correlation-based ensembles

Data set		Average diversity of the base classifiers, (before and after iterations)										Diver-sity diff.	Fea-tures	Aver num of select. feat.	Aver num of itera-tions
Car	CM_b	12.6	12.3	15.9	16.2							7.3	5	4.225	1.5
	CM_a	24.5	20.5	20.2	21.1										
	CR_b	13.6	13.6	17.2	16.8							6.925		3.217	1.6
	CR_a	27.3	23.5	19.3	18.8										
Glass	CM_b	20.8	21.7	24.1	28.2	35.6	32.0					7.5	9	4.611	1.367
	CM_a	33.0	31.6	32.1	35.1	41.4	36.2								
	CR_b	30.4	44.8	31.4	31.3	40.5	28.8					3.52		3.328	1.3
	CR_a	35.6	46.1	35.1	35.3	42.0	34.2								
Iris	CM_b	3.6	1.8	1.9								1.83	4	1.356	1.067
	CM_a	5.7	3.3	3.8											
	CR_b	22.3	44.4	22.3								1.23		1.678	1
	CR_a	24.2	45.2	23.3											
LED_17	CM_b	24.6	39.8	34.5	24.2	24.2	21.8	23.9	30.6	34.2	21.4	5.81	24	22.367	1.5
	CM_a	31.3	44.0	36.5	31.7	32.7	30.5	30.5	35.4	37.0	27.7				
	CR_b	58.3	59.5	73.5	57.7	82.2	66.6	62.3	63.9	55.0	54.1	1.35		4.533	1.6
	CR_a	60.3	60.6	73.9	60.5	81.8	68.5	62.7	66.2	56.7	55.2				
Lymph	CM_b	18.8	15.1	17.7	16.3							7.75	18	6.625	1.6
	CM_a	28.3	22.2	24.0	24.4										
	CR_b	25.1	21.4	21.0	38.2							2.475		5.35	1.267
	CR_a	27.9	24.6	24.2	38.9										
Thyroid	CM_b	6.2	6.4	9.3								2.17	5	1.844	1.4
	CM_a	9.3	8.5	10.6											
	CR_b	14.7	12.4	11.8								1.2		2.944	1.2
	CR_a	15.8	13.8	12.9											
Vehicle	CM_b	24.6	23.1	24.6	24.6							3.875	18	12.258	1.5
	CM_a	29.8	28.7	27.1	27.8										
	CR_b	39.1	43.0	45.5	43.5							0.85		6.492	1.433
	CR_a	40.4	43.7	46.0	44.4										
Wine	CM_b	16.7	15.7	17.5								1.43	12	6.7	1.3
	CM_a	18.8	17.7	17.8											
	CR_b	16.2	18.9	16.6								3.07		3.944	1.3
	CR_a	19.7	21.8	19.4											
Wave-form	CM_b	18.8	21.5	19.7								3.67	21	17.611	1.267
	CM_a	22.8	24.4	23.8											
	CR_b	40.3	39.0	38.8								1.33		8.289	1.467
	CR_a	41.4	40.0	40.1											
Zoo	CM_b	12.6	13.2	12.3	13.9	14.2	13.8	11.9				2.57	16	13.252	1.2
	CM_a	15.7	15.9	14.9	16.3	17.0	15.3	14.8							
	CR_b	37.5	32.9	37.1	44.0	40.3	34.1	30.2				2.66		3.867	1.367
	CR_a	39.5	36.1	39.8	44.3	42.2	36.9	35.9							

In Table 2 for each data set the diversities of all base classifiers are presented. In the sub-columns of the second column diversities of the base classifiers are presented. The number of the base classifiers is equal to the number of classes for each data set. In the third column the average difference of diversity before and after iteration is recorded. The forth column represents the number of features in corresponding data set, and the fifth column show the number of features used to construct the base classifiers. The last column presents the average number of iterations in the refinement cycle.

In order to make conclusions following our aims from data tables above we calculated statistics in Table 3. We applied the paired sign test in order to estimate the number of wins, looses, and ties for each pair of compared classification results, and the paired t-test in order to conclude if the difference was statistically significant. The paired t-test is more strict than the sign test, but it assumes that population follows the Gaussian distribution. We calculated statistics for both tests over 30 runs, and tested if the distribution is normal using standardized skewness and standardized kurtosis.[2] The second and third columns in Table 3 represent statistics for comparison of two heuristic-base ensembles before and after iterations. The forth and fifth columns summarize the difference in accuracy before and after iterations for CM-based and correlation-based ensembles. The sixth and seventh columns represent statistics for comparison of both heuristic-based ensembles after refinement cycle versus the C4.5 algorithm. The sub-columns indicate trial for the sign test, and P value for the paired t-test. Negative results of normal distribution test are outlined. Both statistical tests have been done using 95% confidence interval. Average calculations in the last row of the table have been done over ten data sets using their average accuracy from Table 1.

Table 3. Comparisons of the accuracies obtained by CM-based and correlation-based ensembles, and the accuracies of C4.5 by the sign test and the paired t-test

Data set	CM_b vs CR_b		CM_a vs CR_a		CM_a vs CM_b		CR_a vs CR_b		CM_a vs C4.5		CR_a vs C4.5	
	win, tie, loose	P value 2-tailed	win, tie, loose	P value 2-tailed	win, tie, loose	P value 1-tailed	win, tie, loose	P value 1-tailed	win, tie, loose	P value 2-tailed	win, tie, loose	P value 2-tailed
Car	16,2,12	**0.1052**	19,3,8	0.0136	20,5,5,	0.0001	20,4,6	0.0013	11,6,13	**1.0000**	5,5,20	0.0009
Glass	10,5,15	**0.5249**	12,5,13	**0.4998**	9,14,7	**0.2933**	12,14,4	**0.3048**	19,6,5	0.0156	21,3,6	0.0021
Iris	6,19,5	**0.3219**	5,19,6	**0.8455**	2,26,2	**0.2244**	3,25,2	**0.1862**	6,21,3	**0.3544**	5,23,2	**0.2591**
LED_17	19,7,4	<0.0001	16,4,10	0.0376	8,14,8	**0.4083**	19,9,2	<0.0001	11,5,14	**0.4872**	7,1,22	0.0033
Lymph	14,7,9	**0.0876**	14,6,10	**0.1152**	8,12,10	**0.1669**	6,18,6	**0.2930**	16,4,10	**0.2833**	8,11,11	**0.4999**
Thyroid	7,6,17	0.0018	7,16,7	**0.4171**	16,11,3	0.0011	6,19,5	**0.2339**	10,12,8	**0.7375**	11,13,6	**0.5634**
Vehicle	14,2,14	**0.4682**	12,2,16	**0.2175**	14,7,9	**0.2106**	12,10,8	**0.1732**	8,4,18	0.0138	13,2,15	**0.3385**
Wave-	12,0,18	**0.0524**	13,0,17	**0.9717**	18,8,4	0.0015	9,13,8	**0.6606**	23,2,5	0.0003	23,1,6	0.0008
Wine	13,8,9	**0.2080**	12,6,12	**0.9250**	4,18,8	**0.4976**	8,18,4	**0.0655**	11,10,9	**0.3918**	13,7,10	**0.4832**
Zoo	26,2,2	<0.0001	25,3,2	<0.0001	9,18,3	0.0290	12,14,4	**0.0814**	8,17,5	**0.1653**	2,6,22	<0.000
Average	6,0,4	**0.2356**	6,1,3	**0.1760**	7,1,2	**0.0898**	7,0,3	**0.1719**	6,1,3	**0.1457**	5,0,5	**0.6777**

Trial "win, tie, loose" for example, in comparison of CM-based ensemble before and after iterations (CM_a versus CM_b) for Car data set means that in 20 cases of 30 CM_a has higher accuracy, in 5 cases accuracies are comparable, and in 5 cases accuracy was higher for CM_b. Small P value means that it is unlikely that the effect of iterations is due to a coincidence of random sampling of 30 runs from the overall

[2] Values of these statistics outside the range of –2 to +2 indicate significant abnormality, which would tend to invalidate any statistical test regarding the standard deviation, and the paired t-test in our case.

population. If P value is < 0.05 the observed effect is statistically significant. Large P value in some cases do not give us any reason to conclude that some effect has a place, as far as we can not observe it on current data sample. In order to compare the effect of iterations the one-tailed t-test is used since we can assume that accuracy after refinement tend to be higher than before refinement. Corresponding P value rejects or accepts hypothesis about the effect of the cycle. Comparing heuristic-based ensembles versus C4.5 as well as CM-based ensemble versus correlation-based we cannot predict in advance the accuracy trend. That is why for these cases two-tailed P value is used indicating presence of statistically significant difference in accuracy. Which algorithm outperforms can be seen from the sign test trial.

First, let us compare two heuristic-based ensembles. On average, CM_b outperforms CR_b 6 times according to the sign test. The benefits of CM_b are confirmed as statistically significant for LED_17 and Zoo. CM_b is a few more better than CR_b on Lymph, Wine, and Car. However, according to the t-test for Thyroid superiority of CR_b is statistically significant. On Glass and Waveform CR_b is a few more better as well.[3] Both ensembles CM_b and CR_b, as well as CM_a and CR_a on Iris data set reached the highest accuracy, and the lowest accuracy on LED_17 data set.

After iterations CM_a still outperforms CR_a in 6 cases, in one case they are comparable, and in 3 cases CR_a is better than CM_a. CM_a gives statistically significant superiority in accuracy than CR_a for Car, LED_17 and Zoo, and a few smaller superiority on Lymph data set. CM_b and CM_a are almost comparable on Wine data set.

Thus, we can conclude that CM-based ensembles are more accurate than correlation-based ensembles. However, CR_b in the most of cases produces more diverse initial classifiers, especially for Glass, Iris, LED_17, Thyroid, Vehicle, Waveform, and Zoo. Diversity of CM_b and CR_b is almost comparable for Wine data set only. After iterations diversities smooth out for Glass, Lymph, and Wine. In general, the iterations resulted in larger increase for CM-based ensemble. The advantage of correlation-based ensemble is revealed in small number of features on which the base classifiers are constructed. It is clearly seen for LED_17, Vehicle, Wine, Waveform, and Zoo data sets. The number of iterations is the other important item for comparisons. We can observe that for 5 data sets CM-based ensemble, in average, made more iterations, for 1 data set the average numbers of iterations are equal, and on 4 data sets correlation-based ensemble made more iterations.

Let us analyze now the contribution of iterative refinement cycle intended for increase of diversity and accuracy in ensemble. Diversity was increased over all data sets on 1-7%. Accuracy was increased with both heuristic-based ensembles approximately on 1-2%. For CM-based ensemble the effect of iterations is statistically significant on Car, Thyroid, Waveform, and Zoo. Sometimes overfitting can take place[4], as in this case for Iris and Lymph, and with correlation-based iteration for Thyroid and Waveform data sets. Improvement in accuracy for correlation-based ensemble is

[3] The situation can be found out slightly differing from Table 1 inasmuch as this table includes accuracies averaged over 30 runs.

[4] The accuracies are calculated using the evaluation set, and if the iteration results in overfitting with respect to the test set used during iteration than the final accuracy can become smaller.

statistically significant for Car and LED_17 data sets. For the other data sets we cannot conclude that there is no effect of iterations. It just unobserved on current data.

Finally, we compare heuristic-based ensembles with the C4.5 learning algorithm. Heuristic-based ensembles use much less amount of features in order to construct classifiers, as it is shown in Table 2. The t-test indicated that CM_a is significantly better than C4.5 for Car and Waveform, and C4.5 outperformed on Vehicle. The sign test over all data sets sums up that CM_a works in 6 cases better, in 3 cases worse, and in 1 case tie. The sign test for CR_a shows that CR_a and C4.5 are comparable. CR_a is significantly better for Glass and Waveform data sets, whereas C4.5 is significantly better for Car, LED_17 and Zoo data sets.

6 Conclusions

Ensembles of classifiers can be constructed by a number of methods with the purpose of creating a set of diverse and accurate base classifiers. Feature selection techniques, along with other techniques are applied to prepare the training sets for construction of the base classifiers. In this paper, we have analyzed and experimented with two feature selection heuristics. Both the CM-based and the correlation-based ones rely on quite similar assumptions concerning heterogeneous classification problems. We produced ensembles including as many base classifiers as there are classes and each base classifier was produced by C4.5 to distinguish instances of one class from the other classes. Each classifier is based on a subset of features and these features are selected using the CM-based or correlation-based merit values.

In order to refine the ensemble characteristics, we applied iterative refinement during the final ensemble generation process. The refinement cycle provides feedback promoting more diverse set of base classifiers taking into account global accuracy.

We have evaluated our approach on a number of data sets from the UCI machine learning repository. Experiments showed that CM-based approach often outperformed in accuracy than correlation-based approach, however, the latter in the most of cases produces more diverse classifiers. Iterations have usually greater effect with CM-based approach making the difference in diversity smaller at the end. The correlation-based approach has an advantage producing more simple base classifiers than CM-based because of small number of features used. The iterative refinement cycle increase diversity for CM-based approach more effectively than for correlation-based one. As far as iterations promote more diverse classifiers it seems that such a refinement is more preferable for a heuristic producing the base classifiers of small diversity. Further research is also needed to found more beneficial iterative refinement for the correlation-based approach. CM-based ensemble in many cases works better than C4.5. Correlation-based ensemble is comparable in accuracy with C4.5, at least with current iterative refinement used.

References

1. C., Hong, S.J., Hosking, J.R.M., Lepre, J., Pednault, E.P.D., Rosen, B.K.: Decomposition Apte of heterogeneous classification problems. Advances in Intelligent Data Analysis, Springer-Verlag, London (1997) 17-28.
2. Batitti, R., Colla, A.M.: Democracy in neural nets: voting schemes for classification. Neural Networks, Vol. 7, No. 4 (1994) 691-707.
3. Bauer, E. Kohavi, R.: An empirical comparison of voting classification algorithms: bagging, boosting, and variants. Machine Learning, Vol. 36 (1999) 105-139.
4. Dash, M., Liu, H.: Feature selection for classification. Intelligent Data Analysis, Vol. 1, No. 3, Elsevier Science (1997).
5. Dietterich, T. Machine learning research: four current directions. Artificial Intelligence, Vol. 18, No. 4 (1997) 97-136.
6. Fan, W., Stolfo, S., Chan, P.: Using conflicts among base classifiers to measure the performance of stacking. In: Proc. 16th Int. Conf. on Machine Learning (ICML'99) Workshop on Recent Advanced in Meta-learning and Future Work, (1999) 10-17.
7. Hall, M.: Correlation-based feature selection for discrete and numeric class machine learning. In: Proc. 17th Int. Conf. on Machine learning, Stanford University, CA, Morgan Kaufmann Publishers (2000). [http://www.iit.nrc.ca/bibliographies/feature-selection.html]
8. Hansen, L., Salamon, P.: Neural network ensembles. In: IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. 12 (1990) 993-1001.
9. Hong, S.J.: Use of contextual information for feature ranking and discretization. IEEE Transactions on knowledge and Data Engineering, Vol. 9, No. 5 (1997) 718-730.
10. Howe, N., Cardie C.: Examining locally varying weights for nearest neighbor algorithms. Lecture Notes in Artificial Intelligence, Springer (1997) 455-466.
11. Kohavi, R., Sommerfield, D., Dougherty, J.: Data mining using MLC++: a machine learning library in C++. Tools with Artificial Intelligence, IEEE CS Press (1996) 234-245.
12. Kononenko, I., Simec, E., Robnik, M.: Overcoming the myopia of inductive learning algorithms with RELIEF. Applied Intelligence, Vol. 7 (1997) 39-55.
13. Merz, C.J., Murphy, P.M.: UCI Repository of Machine Learning Data Sets [http://www.ics.uci.edu/ ~mlearn/MLRepository.html]. Dep-t of Information and CS, Un-ty of California, Irvine, CA (1998).
14. Opitz, D., Maclin, R.: Popular ensemble methods: an empirical study. Artificial Intelligent Research, Vol. 11 (1999), 169-198.
15. Opitz, D.: Feature selection for ensembles. In: 16th National Conf. on Artificial Intelligence (AAAI), Orlando, Florida (1999) 379-384.
16. Oza, N., Tumer, K.: Dimensionality Reduction Through Classifier Ensembles. Tech. Rep. NASA-ARC-IC-1999-126.
17. Prodromidis, A. L., Stolfo, S. J., Chan P. K.: Pruning classifiers in a distributed meta-learning system. In: Proc. 1st National Conference on New Information Technologies, (1998) 151-160.
18. Puuronen, S., Skrypnyk, I., Tsymbal, A.: Ensemble feature selection based on the contextual merit. In: Proc. 3rd Int. Conf. on Data Warehousing and Knowledge Discovery (DaWaK'01), (2001) (to appear).
19. Quinlan, J.R.: C4.5: Programs for Machine Learning. Morgan Kaufmann, San Mateo, California (1993).
20. Skrypnyk, I., Tsymbal, A., Puuronen, S.: Local feature selection for heterogeneous problems. In: Proc. 2nd Int. Conf. on Data Mining 2000, WIT Press (2000) 203-212.
21. Tumer, K., Ghosh, J.: Error correlation and error reduction in ensemble classifiers. Connection Science, Vol. 8 Nos. 3,4 (1996) 385-404.

Interactive Constraint-Based Sequential Pattern Mining[*]

Marek Wojciechowski

Poznan University of Technology
Institute of Computing Science
ul. Piotrowo 3a, 60-965 Poznan, Poland
Marek.Wojciechowski@cs.put.poznan.pl

Abstract. Data mining is an interactive and iterative process. It is very likely that a user will execute a series of similar queries differing in pattern constraints and mining parameters, before he or she gets satisfying results. Unfortunately, data mining algorithms currently available suffer from long processing times, which is unacceptable in case of interactive mining. In this paper we discuss efficient processing of sequential pattern queries utilizing cached results of other sequential pattern queries. We analyze differences between sequential pattern queries and propose algorithms that in many cases can be used instead of time-consuming mining algorithms.

1 Introduction

Data mining aims at discovery of useful patterns from large databases or warehouses. One of the most popular data mining methods is sequential pattern discovery introduced in [2]. Informally, sequential patterns are the most frequently occurring subsequences in sequences of sets of items. The initial formulation of the problem was significantly extended in [10], where a taxonomy on items was added to support discovery of so called generalized sequential patterns, and three time constraints (min-gap, max-gap, and time window) were introduced to be used when checking if a given source sequence contains a given pattern. For that extended problem formulation, an efficient algorithm called *GSP* was proposed. Applications of sequential patterns include analysis of telecommunication systems, discovering frequent buying patterns, analysis of patients' medical records, etc.

From a user's point of view, data mining can be seen as an interactive and iterative process of advanced querying: a user specifies the source dataset and the requested class of patterns, the system chooses the appropriate data mining algorithm and returns discovered patterns to the user [4][6]. A user interacting with a data mining system has to specify several constraints on patterns to be discovered. However, usually it is not trivial to find a set of constraints leading

[*] This work was partially supported by the grant no. KBN 43-1309 from the State Committee for Scientific Research (KBN), Poland.

to the satisfying set of patterns. Thus, users are very likely to execute a series of similar data mining queries before they find what they need. Unfortunately, data mining algorithms require long processing times, which makes such interaction difficult.

In this paper, we discuss efficient sequential pattern discovery in the presence of materialized results of previous sequential pattern queries. We claim that a data mining system should exploit the fact that a user is very likely to execute a number of similar sequential pattern queries during a single session. We propose caching results of mining queries by materializing their results on disk (we assume that a data mining system is going to be assigned a certain amount of disk space for that purpose). It is obvious that materialized results of a query can be used to answer an identical query, therefore we concentrate on processing queries different from those whose results are available. The possibility of answering a query using known results of another query depends on the differences between the two queries. Our goal is to provide criteria for determining if cached results of a given query can be used to answer the current query without running a complete mining algorithm, and introduce efficient sequential pattern query processing algorithms exploiting materialized patterns.

Exploiting cached results of previous mining queries has been studied in the context of association rules [3][7]. However, direct application of methods and techniques introduced for association rules to sequential pattern discovery problem is not possible since different types of constraints are available in the two problems. Nevertheless, it seems that the general ideas should stay unchanged.

In has been observed [3] that the three particularly interesting relationships between two mining queries DMQ_1 and DMQ_2 extracting patterns from the same data are equivalence, inclusion, and dominance. The three relationships are interesting since they represent situations, where one data mining query can be efficiently answered using the results of another query. Differences between mining queries leading to these relationships were analyzed only in the context of association rules. In this paper we present analogous analysis concerning sequential patterns. Thus, most of our work can be regarded as the extension of the approach from [3] into sequential pattern discovery.

1.1 Sequential Patterns

Let $L = l_1, l_2, ..., l_m$ be a set of literals called items. An *itemset* is a non-empty set of items. A *sequence* is an ordered list of itemsets and is denoted as $< X_1X_2...X_n >$, where X_i is an itemset ($X_i \subseteq L$). X_i is called an *element* of the sequence. The *size* of a sequence is the number of items in the sequence. The *length* of a sequence is the number of elements in the sequence. Let D be a set of variable length sequences (called *data-sequences*), where for each sequence $S =< X_1X_2...X_n >$, a timestamp is associated with each X_i.

With no time constraints we say that a sequence $X =< X_1X_2...X_n >$ is *contained* in a data-sequence $Y =< Y_1Y_2...Y_m >$ if there exist integers $i_1 < i_2 < ... < i_n$ such that $X_1 \subseteq Y_{i_1}, X_2 \subseteq Y_{i_2}, ..., X_n \subseteq Y_{i_n}$. We call $< Y_{i_1}Y_{i_2}...Y_{i_n} >$ an *occurrence* of X in Y. We consider the following user-specified time constraints while looking for occurrences of a given sequence: minimal and maximal gap allowed between consecutive elements of an occurrence of the sequence (called *min-gap* and *max-gap*), and time window that allows a group of consecutive elements of a data-sequence to be merged and treated as a single element as long as their timestamps are within the user-specified *window-size*.

The *support* of a sequence $< X_1X_2...X_n >$ in D is the fraction of data-sequences in D that contain the sequence. A *sequential pattern* is a sequence whose support in D is above the user-specified threshold.

1.2 Relationships between Results of Data Mining Queries

Two data mining queries are *equivalent* if for all datasets they both return the same set of patterns and the values of statistical significance measures (e.g. support) for each pattern are the same in both cases. A data mining query DMQ_1 *includes* a data mining query DMQ_2 if for all datasets each pattern in the results of DMQ_2 is also returned by DMQ_1 with the same values of the statistical significance measures. A data mining query DMQ_1 *dominates* a data mining query DMQ_2 if for all datasets each pattern in the results of DMQ_2 is also returned by DMQ_1, and for each pattern returned by both queries its values of the statistical significance measures evaluated by DMQ_1 are not less than is case of DMQ_2. Equivalence is a particular case of inclusion, and inclusion is a particular case of dominance. Equivalence, inclusion, and dominance meet the transitivity property.

If for a given query, results of a query equivalent to it, including it, or dominating it are available, the query can be answered without running a costly mining algorithm. In case of equivalence no processing is necessary, since the queries have the same results. In case of inclusion, one scan of the materialized query results is necessary to filter out patterns that do not satisfy constraints of the included query. In case of dominance, one verifying scan of the source dataset is necessary to evaluate the statistical significance of materialized patterns (filtering out the patterns that do not satisfy constraints of the dominated query is also required).

1.3 Related Work

To facilitate interactive and iterative pattern discovery, [8] proposed to materialize patterns discovered with the least restrictive selection criteria, and answer incoming queries by filtering the materialized pattern collection. This approach is not a perfect solution of the problem since pattern mining with very low minimum support thresholds might lead to collections of frequent patterns even larger than the original database. Moreover, restricting certain constraints (e.g. time constraints in the context of sequential pattern mining) not only makes some patterns infrequent but also changes the support of patterns that remain frequent.

Much more reasonable and flexible solutions supporting interactive and iterative mining were presented in [7], in the context of association rules. The solutions presented there consisted in caching results of mining queries. In the approach, materialization of frequent itemsets instead of rules was proposed. However, in some cases it was required to materialize also some of the infrequent itemsets.

Most of the research on sequential patterns focused on introducing new algorithms, more efficient than *GSP* (e.g. [5][9]). However, the novel methods do not handle time constraints and taxonomies. Thus, *GSP* still remains the most general sequential pattern discovery algorithm and the reference point for new methods and techniques.

1.4 Organization of the Paper

The paper is organized as follows. Section 2 presents constraints that can be specified in sequential pattern mining. In Sect. 3, relationships between sequential pattern queries are discussed. Section 4 contains efficient sequential pattern query processing algorithms. Experimental results concerning the proposed algorithms are presented in Sect. 5. We conclude with a summary in Sect. 6.

2 Constraint-Based Sequential Pattern Mining

In constraint-based sequential pattern mining, we identify the following classes of constraints: database constraints, pattern constraints, and time constraints. Database constraints are used to specify the source dataset. Pattern constraints specify which patterns are interesting and should be returned by the query. Finally, time constraints influence the process of checking whether a given data-sequence contains a given pattern.

The basic formulation of the sequential pattern discovery problem introduces three time constraints: max-gap, min-gap, and time window, and assumes only one pattern constraint (expressed by means of the minimum support threshold). We model pattern constraints as complex Boolean predicates having the form of a conjunction of basic Boolean predicates on patterns presented below:

- π(**SPL**, α, pattern) - true if pattern support is less than α, false otherwise;
- π(**SPG**, α, pattern) - true if pattern support is greater than α, false otherwise;
- π(**SL**, α, pattern) - true if pattern size is less than α, false otherwise;
- π(**SG**, α, pattern) - true if pattern size is greater than α, false otherwise;
- π(**LL**, α, pattern) - true if pattern length is less than α, false otherwise;
- π(**LG**, α, pattern) - true if pattern length is greater than α, false otherwise;
- π(**C**, β, pattern) - true if β is a subsequence of the pattern, false otherwise;
- π(**NC**, β, pattern) - true if β is not a subsequence of the pattern, false otherwise.

We believe that the above list of predicates is sufficient to allow users to express their pattern selection criteria. For simplicity's sake, in length and size predicates we consider only sharp inequalities.

3 Relationships between Sequential Pattern Queries

Inclusion and dominance relationships between two data mining queries are defined for queries operating on the same dataset. Therefore, analyzing differences between sequential pattern queries, we consider only differences in time and pattern constraints.

Definition 1. *Given two basic Boolean pattern predicates b_1 and b_2, we say that b_2 is* stronger *than b_1 if one of the following conditions holds:*

1. *$b_1 = \pi(\mathbf{SPG}, \alpha_1, pattern)$ and $b_2 = \pi(\mathbf{SPG}, \alpha_2, pattern)$, where $\alpha_2 > \alpha_1$,*
2. *$b_1 = \pi(\mathbf{SPL}, \alpha_1, pattern)$ and $b_2 = \pi(\mathbf{SPL}, \alpha_2, pattern)$, where $\alpha_2 < \alpha_1$,*
3. *$b_1 = \pi(\mathbf{SG}, \alpha_1, pattern)$ and $b_2 = \pi(\mathbf{SG}, \alpha_2, pattern)$, where $\alpha_2 > \alpha_1$,*
4. *$b_1 = \pi(\mathbf{SL}, \alpha_1, pattern)$ and $b_2 = \pi(\mathbf{SL}, \alpha_2, pattern)$, where $\alpha_2 < \alpha_1$,*
5. *$b_1 = \pi(\mathbf{LG}, \alpha_1, pattern)$ and $b_2 = \pi(\mathbf{LG}, \alpha_2, pattern)$, where $\alpha_2 > \alpha_1$,*
6. *$b_1 = \pi(\mathbf{LL}, \alpha_1, pattern)$ and $b_2 = \pi(\mathbf{LL}, \alpha_2, pattern)$, where $\alpha_2 < \alpha_1$,*
7. *$b_1 = \pi(\mathbf{C}, \beta_1, pattern)$ and $b_2 = \pi(\mathbf{C}, \beta_2, pattern)$, where a pattern β_1 is a subsequence of the pattern β_2 and the size of β_1 is less than the size of β_2,*
8. *$b_1 = \pi(\mathbf{NC}, \beta_1, pattern)$ and $b_2 = \pi(\mathbf{NC}, \beta_2, pattern)$, where pattern β_2 is a subsequence of the pattern β_1 and the size of β_2 is less than the size of β_1.*

Definition 2. *We say that a data mining query DMQ_2* extends pattern constraints *of a data mining query DMQ_1 if any of the following conditions holds:*

1. *Pattern constraints of DMQ_1 have a form of a conjunction of n basic Boolean pattern predicates, pattern constraints of DMQ_2 have a form of a conjunction of $n+1$ basic Boolean pattern predicates ($n \geq 0$), and each basic Boolean pattern predicates in DMQ_1 also appears in DMQ_2;*
2. *DMQ_1 and DMQ_2 have pattern constraints p_1 and p_2 respectively, where p_1 and p_2 are conjunctions of n basic Boolean pattern predicates ($n \geq 1$), $p_1 = p \wedge b_1$, $p_2 = p \wedge b_2$ (p is a conjunction of $n-1$ basic Boolean pattern predicates), and b_2 is stronger than b_1;*
3. *It is possible to formulate a data mining query DMQ_3 such that DMQ_2 extends pattern constraints of DMQ_3 and DMQ_3 extends pattern constraints of DMQ_1. (The relationship of extending pattern constraints is transitive.)*

In other words, a data mining query DMQ_2 extends pattern constraints of a data mining query DMQ_1 if pattern constraints of DMQ_1 can be transformed into pattern constraints of DMQ_2 by appending new basic Boolean pattern predicates or replacing basic Boolean pattern predicates with stronger ones.

Given two sequential pattern queries, there are four cases possible regarding pattern constrains: DMQ_1 and DMQ_2 have the same pattern constraints, DMQ_1 extends pattern constraints of DMQ_2, DMQ_2 extends pattern constraints of DMQ_1, or pattern constraints of DMQ_1 and DMQ_2 are not comparable.

Definition 3. *We say that a data mining query DMQ_2* extends time constraints *of a data mining query DMQ_1 if any of the following conditions holds:*

1. *The value of the max-gap parameter in DMQ_2 is less than in DMQ_1 and both queries have the same value of the min-gap parameter, and the same value of the window-size parameter;*
2. *The value of the min-gap parameter in DMQ_2 is greater than in DMQ_1 and both queries have the same value of the max-gap parameter, and the same value of the window-size parameter;*
3. *The value of the window-size parameter in DMQ_2 is less than in DMQ_1 and both queries have the same value of the max-gap parameter, and the same value of the min-gap parameter;*
4. *It is possible to formulate a data mining query DMQ_3 such that DMQ_2 extends time constraints of DMQ_3 and DMQ_3 extends time constraints of DMQ_1. (The relationship of extending time constraints is transitive.)*

In other words, a data mining query DMQ_2 extends time constraints of a data mining query DMQ_1 if it restricts at least one of the time parameters (max-gap, min-gap, window-size) and does not relax any time parameters.

Given two sequential pattern queries, there are four cases possible regarding time constrains: DMQ_1 and DMQ_2 have the same time constraints, DMQ_1 extends time constraints of DMQ_2, DMQ_2 extends time constraints of DMQ_1, or time constraints of DMQ_1 and DMQ_2 are not comparable.

Example 1. Let us consider the following three sequential pattern queries, operating on the same dataset:

DMQ_1 = {max-gap: 100, min-gap: 0, window-size: 1, π(**SPG**, 0.2, pattern)}
DMQ_2 = {max-gap: 100, min-gap: 0, window-size: 1, π(**SPG**, 0.1, pattern) $\wedge$ π(**SG**, 3, pattern)}
DMQ_3 = {max-gap: 100, min-gap: 7, window-size: 1, π(**SPG**, 0.2, pattern) $\wedge$ π(**SG**, 3, pattern)}

DMQ_3 extends pattern constraints of DMQ_1 and DMQ_2, while pattern constraints of DMQ_1 and DMQ_2 are not comparable. DMQ_3 extends time constraints of DMQ_1 and DMQ_2, while time constraints DMQ_1 and DMQ_2 are the same.

The two relationships defined above concern the syntax of queries, while the general inclusion and dominance relationships refer to results of queries. Below we introduce three theorems regarding dependence of relationships between results of two queries on syntactic differences between the two queries. We also introduce several lemmas on which the proofs of theorems are based. For brevity, we do not include proofs of the lemmas since they come straight from the above definitions and inherent properties of pattern and time constraints.

Lemma 1. *Let b_1 and b_2 be basic Boolean pattern predicates such that b_2 is stronger than b_1. For each pattern p, if p satisfies b_2 then p satisfies b_1.*

Lemma 2. *Let DMQ_1 and DMQ_2 be two sequential pattern queries, operating on the same dataset and having the same time constraints. Let p_1 and p_2 denote pattern constraints of DMQ_1 and DMQ_2 respectively. If $p_2 = p_1 \wedge b$, where b is a basic Boolean pattern predicate, then DMQ_1 includes DMQ_2.*

Lemma 3. *Let DMQ_1 and DMQ_2 be two sequential pattern queries, operating on the same dataset and having the same time constraints. Let p_1 and p_2 denote pattern constraints of DMQ_1 and DMQ_2 respectively. If $p_1 = p \wedge b_1$ and $p_2 = p \wedge b_2$, where p is a conjunction of n basic Boolean pattern predicates ($n \geq 0$) and b_2 is stronger than b_1, then DMQ_1 includes DMQ_2.*

Theorem 1. *Let DMQ_1 and DMQ_2 be two sequential pattern queries, operating on the same dataset and having the same time constraints. If DMQ_2 extends pattern constraints of DMQ_1, then DMQ_1 includes DMQ_2.*

Proof. From the Definition 2, we know that if DMQ_2 extends pattern constraints of DMQ_1, then it is possible to formulate a sequence of sequential pattern queries DMQ_{i_1}, DMQ_{i_2}, ..., DMQ_{i_n} operating on the same dataset and having the same time constraints as DMQ_1 and DMQ_2, such that $DMQ_{i_1} = DMQ_1$ and $DMQ_{i_n} = DMQ_2$, and for $j = 2..n$ one of the following conditions holds:

1. pattern constraints of $DMQ_{i_{j-1}}$ have a form of a conjunction of n basic Boolean pattern predicates, pattern constraints of DMQ_{i_j} have a form of a conjunction of $n + 1$ basic Boolean pattern predicates ($n \geq 0$), and each basic Boolean pattern predicates in $DMQ_{i_{j-1}}$ also appears in DMQ_{i_j};
2. $DMQ_{i_{j-1}}$ and DMQ_{i_j} have pattern constraints p_1 and p_2 respectively, where p_1 and p_2 are conjunction of n basic Boolean pattern predicates ($n \geq 1$), $p_1 = p \wedge b_1$, $p_2 = p \wedge b_2$ (p is a conjunction of $n - 1$ basic Boolean pattern predicates), and b_2 is stronger than b_1.

From the Lemmas 2 and 3 and the transitivity property of the inclusion relationship, we have DMQ_1 includes DMQ_2.

Lemma 4. *Let DMQ_1 and DMQ_2 be two sequential pattern queries, operating on the same dataset and having the same pattern constraints. Let max_1, min_1, and win_1 denote values of max-gap, min-gap, and window-size parameters of DMQ_1, and max_2, min_2, and win_2 values of max-gap, min-gap, and window-size parameters of DMQ_2. If one of the following conditions holds:*

1. *$max_2 < max_1$ and $min_2 = min_1$ and $win_2 = win_1$,*
2. *$min_2 > min_1$ and $max_2 = max_1$ and $win_2 = win_1$,*
3. *$win_2 < win_1$ and $max_2 = max_1$ and $min_2 = min_1$*

then DMQ_1 dominates DMQ_2.

Theorem 2. *Let DMQ_1 and DMQ_2 be two sequential pattern queries, operating on the same dataset and having the same pattern constraints. If DMQ_2 extends time constraints of DMQ_1, then DMQ_1 dominates DMQ_2.*

Proof. Let max_1, min_1, and win_1 denote values of max-gap, min-gap, and window-size parameters of DMQ_1, and max_2, min_2, and win_2 values of max-gap, min-gap, and window-size parameters of DMQ_2. Since DMQ_2 extends time constraints of DMQ_1, we have: $win_2 \leq win_1$, $max_2 \leq max_1$ and $min_2 \geq min_1$. Let DMQ_3 and DMQ_4 be sequential pattern queries operating on the same dataset and having the same pattern constraints as DMQ_1 and DMQ_2, Let the values of max-gap, min-gap, and window-size parameters be max_2, min_1, and win_1 in case of DMQ_3, and max_2, min_2, and win_1 in case of DMQ_4. Thus, from the Lemma 4, DMQ_1 dominates DMQ_3, DMQ_3 dominates DMQ_4, and DMQ_4 dominates DMQ_2 (in fact, in each of the three cases equivalence is possible but equivalence is a particular case of dominance). Since the dominance relationship is transitive, DMQ_1 dominates DMQ_2.

Theorem 3. *Let DMQ_1 and DMQ_2 be two sequential pattern queries, operating on the same dataset. If DMQ_2 extends pattern constraints of DMQ_1 and DMQ_2 extends time constraints of DMQ_1, then DMQ_1 dominates DMQ_2.*

Proof. Let DMQ_3 be a sequential pattern query operating on the same dataset as DMQ_1 and DMQ_2, having pattern constraints of DMQ_1 and time constraints of DMQ_2. Thus, DMQ_2 extends pattern constraints of DMQ_3 and DMQ_3 extends time constraints of DMQ_1. From the Theorems 1 and 2 we have: DMQ_1 dominates DMQ_3 and DMQ_3 includes DMQ_2. Since inclusion is a particular case of dominance and the dominance relationship is transitive, DMQ_1 dominates DMQ_2.

4 Algorithms for Efficient Sequential Pattern Query Processing in the Presence of Materialized Results of Previous Queries

Given a sequential pattern query DMQ and materialized results of a sequential pattern query DMQ_V, in the general case, even if DMQ_V and DMQ operate on the same dataset but differ in pattern and time constraints, it is not possible to answer DMQ without running a sequential pattern mining algorithm. However, there are four particular cases where DMQ can be answered efficiently using the materialized results of DMQ_V since they correspond to equivalence, inclusion, and dominance relationships between DMQ_V and DMQ. These cases are listed below:

1. If DMQ_V and DMQ have the same pattern and time constraints, then the results of DMQ are equal to the results of DMQ_V (the two queries are equivalent since they are identical);
2. If DMQ_V and DMQ have the same time constraints and DMQ extends pattern constraints of DMQ_V, then DMQ can be answered by filtering out the patterns returned by DMQ_V not satisfying pattern constraints of DMQ (DMQ_V includes DMQ according to the Theorem 1);
3. If DMQ_V and DMQ have the same pattern constraints and DMQ extends time constraints of DMQ_V, then DMQ can be answered by evaluating the support of the patterns returned by DMQ_V using the time constraints of DMQ, and filtering out patterns not satisfying the minimum support threshold of DMQ. (DMQ_V dominates DMQ according to the Theorem 2);

4. If DMQ extends pattern constraints of DMQ_V and DMQ extends time constraints of DMQ_V, then DMQ can be answered by evaluating the support of the patterns returned by DMQ_V using the time constraints of DMQ, and filtering out patterns not satisfying the pattern constraints of DMQ. (DMQ_V dominates DMQ according to the Theorem 3).

Answering the query in the first case (the case of equivalence) is trivial, therefore we concentrate on details concerning inclusion and dominance relationships.

For the second case we propose an algorithm that performs one sequential scan of the materialized patterns, processing one pattern at a time (main memory requirements are minimal). Each pattern is tested if it satisfies these basic Boolean pattern predicates from the pattern constraints of DMQ that were not in DMQ_V. All the basic Boolean pattern predicates of DMQ that were in DMQ_V must be satisfied by all the materialized patterns since pattern constraints in our model have the form of a conjunction of basic predicates. The algorithm for the second case is presented below.

Algorithm 1 Answering a sequential pattern query in case of inclusion due to extending pattern constraints (Result Filtering)
Input: A sequential pattern query issued by a user (DMQ) and results of a sequential pattern query DMQ_V including DMQ.
Output: The results of DMQ.
Method:

```
begin
    $Answer$ = results of $DMQ_V$;
    for each $p \in$ results of $DMQ_V$ do
    begin
        for each basic Boolean pattern predicate $b$ such that
            $b$ is in pattern constraints of $DMQ$ and
            $b$ is not in pattern constraints of $DMQ_V$ do
            begin
                if not ($p$ satisfies $b$) then
                    $Answer = Answer \setminus \{p\}$;
                    break;
                end if;
            end;
    end;
    output $Answer$;
end.
```

For the third and fourth cases we propose one uniform algorithm (both cases result in the dominance relationship). Conceptually, the algorithm has to scan the source dataset once in order to re-evaluate the support of materialized patterns and then prune the patterns that do not satisfy pattern constraints of DMQ. However, for the fourth case, we apply one optimization to reduce the cost of the support re-evaluation phase that is proportional to the number of patterns to be verified. Before scanning the source dataset, we filter out patterns that do not satisfy pattern constraints of DMQ using Algorithm 1. After the scan of the dataset, we only test the predicate representing the minimum support threshold (the only one that for a given pattern could by true before the support re-evaluation, and false after that operation). The effects of this optimization will be discussed in the next section.

During the support re-evaluation phase, when testing whether a currently processed data-sequence contains a given pattern, all time constraints of DMQ have to be taken into account, even if only one of them has been restricted compared to DMQ_V. This is motivated by the observation that a given pattern may occur several times in a given data-sequence. As a result, if we checked only one of the time constraints, we might find a different occurrence satisfying the constraint than the occurrence previously found as valid with respect to the other two time constraints.

The algorithm in the form presented below assumes that the set of materialized patterns supporting pattern constraints of DMQ fits into main memory. If this is not the case, the set of materialized patterns has to be partitioned into portions that fit into main memory and the algorithm has to be run on each of the partitions.

Algorithm 2 Answering a sequential pattern query in case of dominance due to extending time constraints (Result Verification)
Input: A sequential pattern query issued by a user (DMQ), a collection of data-sequences D, and results of a sequential pattern query DMQ_V dominating DMQ.
Output: The results of DMQ.
Method:

```
begin
    if $DMQ$ extends pattern constraints of $DMQ_V$ then
        $Answer$ = patterns in results of $DMQ_V$ satisfying
        pattern constraints of $DMQ$; /* Algorithm 1 */
    else $Answer$ = results of $DMQ_V$;
    end if;
    scan $D$ once evaluating the support of patterns
    in $Answer$ using time constraints of $DMQ$;
    for each $p \in Answer$ do
    begin
        if $p$ exceeds the minimum support threshold of $DMQ$
        then output $p$; end if;
    end;
end.
```

Having provided sequential pattern query processing algorithms for the cases leading to equivalence, inclusion and dominance relationships, we have to address situations where for a given query issued by a user (DMQ), there are many materialized query results that could be used to answer the query without running a complete data mining algorithm. In general, the set of applicable materialized query results consists of results of queries equivalent to DMQ, including DMQ, and dominating DMQ. It is clear that in the first place the data mining system should look for a query identical to DMQ (the case of equivalence) since in that case the results of DMQ are directly available. Then, the system should look for query results that could be used by Algorithm 1 (returned by a query DMQ_V having the same time constraints as DMQ, such that DMQ extends pattern constraints of DMQ_V). If no query satisfying the above criteria could be found, the system should try to find query results that could be used by Algorithm 2 (returned by a query DMQ_V, such that DMQ extends time constraints of DMQ_V and either DMQ_V and DMQ have the same pattern constraints or DMQ extends pattern constraints of DMQ_V). Finally, if again no appropriate query criteria could be found, a complete data mining algorithm has to be run.

We believe that in majority of cases Algorithm 1 will be more efficient than Algorithm 2 since the former requires one scan of the pattern set and no scan of the source dataset, while the latter scans the source dataset once and during this scan for each data-sequence processes all the patterns. However, it has to be noted that in certain cases application of Algorithm 2 may be more efficient than application of Algorithm 1 (for example, if the source dataset and the materialized set of patterns to be used by Algorithm 2 are extremely small, whereas the materialized pattern set to be used by Algorithm 1 is huge).

The final issue that has to be addressed is the selection of the materialized query results to be used by Algorithms 1 and 2 if there is more than one query including or dominating the query to be answered. We observe that it is not possible to provide selection criteria always leading to the minimal processing time, because the processing time depends not only on the syntax of the queries but also on the contents of the source dataset. Therefore, we decide to optimize the space requirements by choosing the materialized pattern set of the smallest size. We believe that this solution will also lead to minimal processing time in many situations, since smaller size of the pattern set leads to the smaller number or size of patterns that have to be filtered or verified against the database. It is not guaranteed, however, since the processing time is affected also by the number of predicates that have to be tested for each pattern, which depends on the pattern structure (subsequent predicates are tested until one of them is found to be false).

Example 2. Let us consider the following three queries discovering sequential patterns from the same dataset:

> DMQ_1 = {max-gap: 100, min-gap: 0, window-size: 1, π(**SPG**, 0.2, pattern)}
> DMQ_2 = {max-gap: 100, min-gap: 7, window-size: 1, π(**SPG**, 0.1, pattern) $\wedge$ π(**SG**, 3, pattern)}
> DMQ = {max-gap: 100, min-gap: 7, window-size: 1, π(**SPG**, 0.2, pattern) $\wedge$ π(**SG**, 3, pattern)}

Let us assume that results of DMQ_1 and DMQ_2 are stored in cache, and DMQ is the query to be answered. Since neither DMQ_1 nor DMQ_2 is identical to

DMQ, the data mining system would choose to answer DMQ using Algorithm 1 exploiting cached results of DMQ_2 (returning those patterns from the results of DMQ_2 that exceed the minimum support threshold of 0.2). If results of DMQ_2 were not available, the system would answer DMQ using Algorithm 2 exploiting the results of DMQ_1 (selecting patterns from the results of DMQ_1 whose size is greater than 3, re-evaluating their support in one scan of the source dataset using max-gap of 100, min-gap of 7, and window-size of 1, and returning those patterns that exceed the minimum support threshold of 0.2 after support re-evaluation).

5 Experimental Results

In order to evaluate performance gains offered by our sequential pattern query processing algorithms, we performed several experiments on a synthetic dataset generated by means of the *GEN* generator from the *Quest* project [1]. We treated transaction identifiers generated by *GEN* as transaction times. Thus, the time gap between two adjacent elements of each data-sequence was always equal to one time unit. The dataset used in the experiments consisted of 1000 data-sequences. *GEN* parameter values were chosen so that for the minimum support thresholds used in queries there were a reasonable number of sequential patterns varying in size and length to be discovered.

In the first step we materialized the results of the query discovering all sequential patterns whose support was above 0.5% using max-gap of 1000, min-gap of 0, and window-size of 1. The materialized set of patterns consisted of about 3500 sequential patterns. Next, we tested several queries adding additional pattern constraints (concerning pattern support, size, length, or contents) and restricting time constraints. For each query, we compared execution times of our algorithms exploiting materialized patterns and the *GSP* algorithm with the post-processing pattern filtering phase. For the queries included by the materialized query, Algorithm 1 was on average more than 400 times faster than *GSP*. For the queries dominated by the materialized query, Algorithm 2 was used, and its processing time was on average more than 100 times shorter than in case of *GSP*. We also tested the effects of our optimization used in case of queries extending both pattern and time constraints of the materialized query (filtering out patterns that do not satisfy pattern constraints before re-evaluating the support of materialized patterns). Experiments show that the optimization reduces processing time by about 33%.

6 Concluding Remarks

We proved experimentally that our sequential pattern query processing schemes can reduce processing time by several orders of magnitude when materialized results of previous queries are available. However, theoretically it is possible to imagine situations, where a complete mining algorithm could be more efficient than our techniques. While we believe that in typical situations our methods should outperform mining algorithms, in the future we plan to focus on cost-based optimization of sequential pattern queries (and data mining queries in general) using certain statistics of the source dataset in order to choose an optimal query execution plan.

In the paper, we did not discuss cache management schemes, which could certainly influence the overall performance of the system. We believe that general purpose cache management algorithms could be used, possibly with simple optimizations such as removing included or dominated queries first, and not materializing results of queries equivalent to queries whose results are already in cache.

References

1. Agrawal R., Mehta M., Shafer J., Srikant R., Arning A., Bollinger T.: The Quest Data Mining System. Proc. of the 2nd KDD Conference (1996)
2. Agrawal R., Srikant R.: Mining Sequential Patterns. Proc. of the 11th ICDE Conf. (1995)
3. Baralis E., Psaila G.: Incremental Refinement of Mining Queries. Proc. of the 1st DaWaK Conference (1999)
4. Han J., Lakshmanan L., Ng R.: Constraint-Based Multidimensional Data Mining. IEEE Computer, Vol. 32, No. 8 (1999)
5. Han J., Pei J., Mortazavi-Asl B., Chen Q., Dayal U., Hsu M-C.: FreeSpan: Frequent Pattern-Projected Sequential Pattern Mining. Proc. of the 6th KDD Conference (2000)
6. Imielinski T., Mannila H.: A Database Perspective on Knowledge Discovery. Communications of the ACM, Vol. 39, No. 11 (1996)
7. Nag B., Deshpande P.M., DeWitt D.J.: Using a Knowledge Cache for Interactive Discovery of Association Rules. Proc. of the 5th KDD Conference (1999)
8. Parthasarathy S., Zaki M.J., Ogihara M., Dwarkadas S.: Incremental and Interactive Sequence Mining. Proc. of the 8th CIKM Conference (1999)
9. Pei J., Han J., Mortazavi-Asl B., Pinto H., Chen Q., Dayal U., Hsu M-C.: PrefixSpan: Mining Sequential Patterns Efficiently by Prefix-Projected Pattern Growth. Proc. of the 17th ICDE Conference (2001)
10. Srikant R., Agrawal R.: Mining Sequential Patterns: Generalizations and Performance Improvements. Proc. of the 5th EDBT Conference (1996)

Evaluation of a Broadcast Scheduling Algorithm

Murat Karakaya[1] and Özgür Ulusoy[2]

[1] Department of Technical Sciences
Turkish Land Forces Academy, Ankara 06100, Turkey
[2] Department of Computer Engineering
Bilkent University, Ankara 06533, Turkey
muratk@kho.edu.tr, oulusoy@cs.bilkent.edu.tr

Abstract. One of the two main approaches of data broadcasting is *pull-based* data delivery. In this paper, we focus on the problem of scheduling data items to broadcast in such a pull-based environment. Previous work has shown that the *Longest Wait First* heuristic has the best performance results compared to all other broadcast scheduling algorithms, however the decision overhead avoids its practical implementation. Observing this fact, we propose an efficient broadcast scheduling algorithm which is based on an approximate version of the Longest Wait First heuristic. We also compare the performance of the proposed algorithm against well-known broadcast scheduling algorithms.

1 Introduction

There exist two main approaches for data dissemination in broadcast systems: *push* and *pull* [1,2,12,19]. In push-based data delivery, the information server tries to predict data needs using the knowledge provided by user profiles or subscriptions. The server constructs a broadcast schedule in which initiation of data transmission does not require an explicit request from mobile users. The server repetitively transmits the content of broadcast schedule to user population. Mobile users monitor the broadcast channel and retrieve the items they require as they arrive. On the other hand, in a pull-based environment, clients explicitly request data items by sending message to the server. The requests are compiled in a service queue, and a scheduling algorithm decides which data item should be broadcast.

The main contributions of our work can be described as follows. First, previous work has shown that the *Longest Wait First* (LWF) heuristic has the best performance results compared to all other broadcast scheduling algorithms, however the decision overhead avoids its practical implementation [6,7,19,5]. We propose to use an approximate version of the LWF heuristic which can considerably remove the decision overhead of LWF. Second, the implementation of the approximate heuristic is carefully designed and also parameterized to increase the performance with respect to different criteria. Third, some heuristics proposed to be used in push-based broadcast environments are modified and evaluated in the pull-based broadcast environment we simulate. And, finally detailed simulation tests are conducted and reported.

The remainder of the paper is organized as follows. In Sect. 2, we introduce a mobile computing environment that we assume in our work and summarize related work. In Sect. 3, we describe our approximate heuristic and its implementation, *Bucketing* scheduling algorithm. Performance evaluation results of the proposed algorithm are provided and compared with the results of some well-known broadcast scheduling algorithms in Sect. 4. Finally, in Sect. 5 concluding remarks are provided.

2 Background

2.1 Mobile Computing Environment

In a common architectural model used for a mobile computing environment [8,9,12], geographical area is divided into regions, called *cells*, each of which is covered and serviced by a stationary controller. There exist two types of computers; *mobile units* (computers) (MUs) and *stationary computers* (SCs). SCs are connected together via a fixed network. Some of SCs are equipped with wireless interfaces to communicate with MUs and called *mobile support stations* (MSSs). MSSs behave as entry points from MUs to the fixed network. MUs can consume and also produce information by querying and updating the online database stored on SCs. MSSs can be proxy servers on behalf of the other SCs or they can themselves be information servers.

It is assumed in this mobile environment that there is a single broadcast channel dedicated to data broadcast. Users monitor this channel continuously to get the data items they require. There is a *backchannel* which enables MUs to send data requests to MSSs.

2.2 Related Work

The first work related with broadcasting in a pull-based environment is by Ammar and Wong in the context of *teletext* and *videotext* systems [6,7,19]. In [19], Wong proposes three alternative architectures for broadcast information delivery systems: *one-way broadcast (push)*, *two-way interaction (pull)*, and *one-way broadcast/two-way interaction (hybrid).* The heuristics used in two-way interaction are as follows [19]:

– The well-known FCFS algorithm has been modified in such a way that if a page has been requested and placed in the service queue, a new request for that page is ignored. In this way, redundant broadcasts of the same page are avoided [5].
– Another heuristic proposed to be used in broadcast scheduling is *Most Requested First* (MRF). As the name of the heuristic implies, the page with the largest number of pending requests is selected to broadcast.

– The MRF heuristic is configured to break ties in favor of the page with the lowest request probability if the request probabilities of the pages are available to the scheduling algorithm. This version of the heuristic is termed *Most Requested First Lowest* (MRFL).
– The heuristic which selects the page with the largest total waiting time of all pending requests is *Longest Wait First* (LWF).

These heuristics are evaluated in [19] and it is concluded that when the system load is light, the mean response time is not sensitive to the heuristic used. This is due to the fact that in light loads, few scheduling decisions need to be made. On the other hand, when the system load is high and the page request probabilities follow *Zipf's Law* [20], LWF has the best performance, whereas FCFS has the worst.

Vaidya et al. have worked on data broadcast scheduling algorithms for push-based environments extensively and proposed several scheduling algorithms [10, 11,13,14,17,18]. In [13], Jiang and Vaidya also investigate how the variance of response time can be minimized. The authors claim that their work and algorithms can be applied to pull-based environments as well. Therefore, we take their algorithms into consideration while devising our heuristic.

The work which is most related to our work is the one performed by Aksoy and Franklin [4,5]. The authors have proposed a scheduling algorithm which improves and unifies FCFS and MRF heuristics. They conclude that the LWF heuristic has the best performance results according to overall mean waiting time. However, the authors also point out that the straightforward implementation of LWF is not practical. Aksoy and Franklin suggest to integrate FCFS and MRF in a practical way to combine their advantages and eliminate the disadvantages. As a result, the authors propose the RxW heuristic which balances the selection criterion between the number of pending requests and the first request arrival time of a data item. RxW computes the product of the total number of pending requests (R) and waiting time of the first request (W) of that data item, and selects the data item with the maximum RxW value.

3 Bucketing Algorithm

We have aimed to develop a scheduling algorithm that can minimize both the mean waiting time and its variance, as well as is robust to changes in mobile environment and has lower overhead. We describe a new heuristic that we name *Approximate Total Waiting Time* (ATWT). The proposed ATWT heuristic is implemented using a bucketing scheme and the resulting algorithm is termed *Bucketing Algorithm.*

3.1 Approximate Total Waiting Time

In order to decrease the amount of computation, we first assume that all requests for a page come at the same time as the first one. Thus, we only keep the arrival time of the first request for each page. When we need to compute total waiting time of a page, we can simply multiply the number of pending requests with the elapsed time since the first request arrived. This approximation gives us the upper bound of total waiting time of a page. Provided that requests arrival is governed by the Poisson process, if a page is broadcast τ time units after the arrival of the first request to it, mean waiting time for pending requests for this page is $\frac{\tau}{2}$ [6,17]. This fact gives an approximation to compute total waiting time for a page as follows:

$$W_p(t) = \frac{t - A_p}{2} * R_p(t) \tag{1}$$

where t is the current time, A_p is the first request arrival time, and $R_p(t)$ is the total number of pending requests for page p at time t. $W_p(t)$ is the approximate total waiting time for page p. The LWF heuristic needs to compute the total waiting time for every page to select the one with the largest value. We can drop the division by 2 in (1) to simplify the calculation since $W_p(t)$ of each page will be compared. This finalizes the basic formulation, that we call *Approximate Total Waiting Time* (ATWT). ATWT enables us to record less information and do less computation[1].

Finding the Maximal ATWT. The direct implementation of the heuristic we propose above has a time complexity of $O(N)$, where N is the total number of requested pages. In order to avoid the calculation of each requested page's ATWT, we use a method which selects a few pages and calculates only their ATWTs to select the page with *maximal ATWT* value. Our implementation is based on a bucketing technique. We classify the pages according to the number of pending requests associated with them. All the pages that lie in bucket i will have pending request numbers ranging between 2^{i-1} and 2^i-1. The number of buckets is limited by the number of pending requests for distinct pages. There will be $\lceil \log(R+1) \rceil$ buckets of pages, where R is the number of pending requests of the most requested page in the system. In each bucket, the pages are ordered according to their first request arrival time. The first page of each bucket is the first requested page within that bucket.

Whenever we need to find the page with the maximal ATWT, we compare only the ATWT values of the first pages of each bucket. Since the number of buckets is logarithmic with respect to the most requested page's request number, we would examine very few candidates. The page with the largest ATWT value is selected among the first entries of all buckets. It can be shown that the bucketing scheme results in selecting a page with an ATWT value which is at least half of the maximum ATWT value.

[1] A formula similar to the approximation we provide for total waiting time has also been suggested by Aksoy and Franklin [5] but through completely different reasoning and observations from ATWT.

3.2 Implementation of Bucketing Algorithm

The data structure used for each requested page in Bucketing algorithm is illustrated in Fig. 1. Each bucket is a linked list of requested pages[2]. Pages are ordered in the linked lists according to the first request arrival time. Fields *Prev* and *Next* are pointers to the previous and next pages, respectively in the linked list.

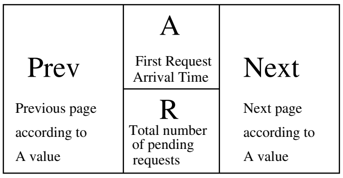

A
First Request Arrival Time
Prev
Previous page according to A value
R
Total number of pending requests
Next
Next page according to A value

Fig. 1. Page data structure

Entries for pages are placed in buckets by mapping total number of requests to bucket number. A page with a total request number i is placed to bucket $\lfloor \log(i) \rfloor + 1$.

Bucketing algorithm works as follows: when a request arrives to the server, if it is the first request for the page, its arrival time is recorded to field A of the page data structure and the number of pending requests (R) is set to one. The page is placed at the end of the linked list in the first bucket since its R value is 1.

Otherwise, if the page was requested and not yet broadcast, R is incremented by one. Then, if the page does not belong to the existing bucket anymore, it is moved to the appropriate bucket according to its R value. The page is then inserted in the linked list of this bucket with respect to its A value.

In the selection of the page to broadcast, only the first page of each bucket is examined. The page with the largest ATWT is broadcast and removed from the bucket. The bucketing scheme reduces the decision overhead considerably without deteriorating the quality of the produced broadcast.

We have also implemented a variant of Bucketing algorithm, called *k-depth* Bucketing algorithm, in which we examine the first k entries of each bucket. By comparing more entries in a bucket, it is expected to have more accurate ATWT.

Minimizing the Variance of Waiting Time. We have investigated the variance of waiting time produced by ATWT and several other heuristics. In [13], Jiang and Vaidya reformulate the algorithm presented in [17] considering variance metric and propose a new algorithm called *α-algorithm*. The authors also

[2] For performance concerns, instead of using a linked list data structure, a heap data structure can also be implemented to store the items in each bucket. However, for the sake of simplicity we prefer to implement a linked list data structure in the simulation.

claim that the algorithm can be adapted to pull-based systems as well. Therefore, we modify the computation of ATWT in our heuristic in a way similar to that suggested in [13] as follows:

$$(t - A_p)^{\alpha} * R_p(t) \tag{2}$$

where α can be assigned different values in order to tune variance of waiting time and mean waiting time.

4 Simulation Results

We have simulated the mobile environment introduced in Sect. 2.1. The simulation program was written in CSIM [16]. Due to lack of space we can not provide all the experiments and their results. For more details please refer to [15].

4.1 Simulation Model

Our simulation model consists of three main components: a *mobile support station* (MSS), a population of *mobile units* (MUs) and *communication channels.* Published requests are kept in *service queue. Online database* stores the shared data items. The decision process is performed by a *scheduling algorithm.* Client population represents MUs within the cell. Communication channel is a two-way medium. In *broadcast channel*, selected data items are delivered to MUs, whereas *backchannel* is used to send data requests of MUs to MSS.

Simulation parameters and their values are summarized in Table 1. *dbSize* is the total number of available data items at an *MSS*. Data items are numbered from 1 to *dbSize*, where a data item is, for example, a web page or a file. We use the terms *data item* and *page* interchangeably since the information server can be a database or web server.

Table 1. Simulation parameters

Symbol	Description	Default	Range	Unit
λ	Mean Req.Arrival Rate	10	[10-100]	req./tick
Θ	Request Pattern Skewness	1.0	[0.1-1.0]	-
dbSize	Database Size	1,000	[1,000-10,000]	pages
pSize	Page Size	1	-	tick

Requests of MUs are represented by a single request stream. Request arrivals are assumed to be *Poisson* with a mean value of λ. By increasing λ, we can simulate a higher system load. MUs may exhibit data locality, querying a particular subset of the database repeatedly [9,12]. This subset is a *hot spot* for an MU. In general, a user may request multiple items simultaneously and would expect to receive mutually consistent versions of the requested items. In this paper, similar to many of the past work, we consider the case where a user demands only one item per request, and unless the user gets the item, a new request is not initiated. In our work, the effect of transmission errors is not considered. We assume that when a data item is broadcast, all the users requesting that item receive it completely. It is assumed that access probabilities follow the *Zipf* [20] distribution over the database items as in many other related work (e.g., [3,5, 19]). Data items are supposed to be ordered in the database according to their access probabilities in decreasing order, i.e., the most favorite data item is in the first place in the database. Zipf's law states that the relative probability of a request for the *i*'th most popular data item is proportional to $\frac{1}{i}$, where i is between 1 and dbSize. The Zipf distribution can be formulated to show the demand probability of each data item as below:

$$p_i = \frac{(1/i)^{\Theta}}{\Sigma_{i=1}^{dbSize}(1/i)^{\Theta}} \tag{3}$$

where Θ is a parameter termed *access skew coefficient* [18]. By changing the value of Θ, different Zipf distributions can be obtained.

The time to broadcast a data item is calculated by a specific time unit called *tick*. We assume that page sizes (*pSize*) are equal and each page can be transmitted in one tick. The use of tick as a time unit enables us to compare easily the results of systems with different properties such as bandwidth and data item size. For evaluating a broadcast scheduling algorithm for a particular set of parameters, the broadcast schedule is produced for at least 30,000 cycles. Furthermore, we run each configuration ten times and use the averages as final estimates.

4.2 Performance Criteria

We have used the following performance criteria in our evaluations:

– *Waiting time of a request* is defined as the duration from when the request is made until the desired data item begins to be transmitted on the channel.
– *Variance of waiting time* is taken into consideration to evaluate the Quality of Service experienced by any user [13], where the overall mean waiting time is an indication of the idle time for the whole user population.
– *Worst waiting time* is defined as the maximum amount of time that any user request waits before being satisfied. The reason to use this criterion is to check if the algorithm causes starvation of some requests, which is an important property for interactive applications [5].
– *Decision overhead* is the time taken by the computation which should be done for selecting a data item to broadcast next. The decision overhead of a good scheduling algorithm should not be high.

4.3 Mean Waiting Time

In this experiment, mean request arrival rate (λ) is varied from 10 requests to 100 requests per tick. As depicted in Fig. 2, mean waiting times for all algorithms are increasing while the request arrival rate is getting higher. However, after a certain rate, it levels off. In the figure, the results of MRF and FCFS are not presented. Because they have much larger mean waiting time than the other algorithms. LWF, RxW, and Bucketing algorithms are characterized by almost the same mean waiting time. The largest difference between any two of these three algorithms is not more than 0.8%.

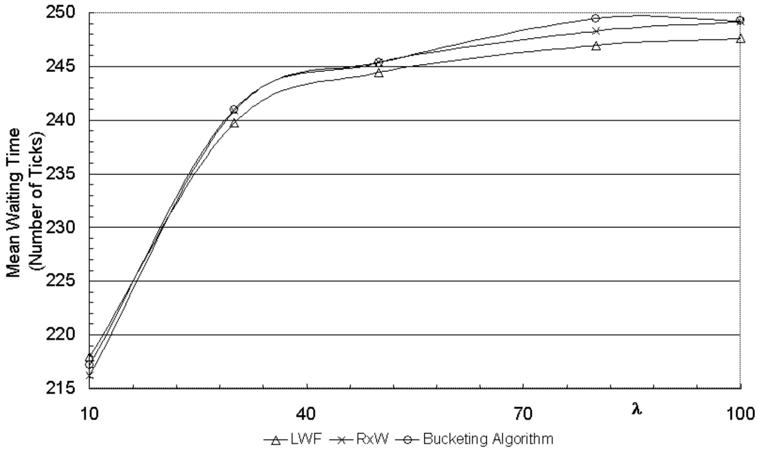


Fig. 2. Mean waiting times of the LWF, RxW, and Bucketing algorithms

Although LWF is a good algorithm in terms of mean waiting time, straightforward implementation of it is not practical for large databases and high-speed broadcast channels. RxW and Bucketing algorithm are the only algorithms which are practical to implement and satisfactory in terms of mean waiting time results.

We have conducted several simulation tests to record the impact of the different database sizes and access skewness values. It is observed that the resulting waiting times of algorithms are relatively almost the same. As the database size increases, the mean waiting time increases as well. There is not much difference between the scalability of the algorithms with respect to database size. Therefore, concerning the time required to perform the simulation experiments, we preferred to use a default database size of 1,000 pages for all other experiments without losing generality. As the skewness of the Zipf distribution is increased, mean waiting time values of the algorithms, except FCFS, are getting considerably smaller. This result is due to the fact that the highly skewed request distribution (i.e., $\theta \geq 0.7$) leads to the existence of many pending requests to a few data items, and that broadcasting one of the most requested data items satisfies many pending requests. RxW, LWF and Bucketing algorithms take the number of pending requests into account and this property causes more efficient use of the broadcast channel. However, when θ is 0.1, the distribution reduces to an almost uniform distribution, and each data item has almost the same pending requests. In that case, all the scheduling algorithms lead to almost the same mean waiting time. FCFS does not consider the pending request number in broadcast scheduling. Hence, the mean waiting time obtained with this algorithm does not improve much when the access skewness increases.

4.4 Variance of Waiting Time

In Sect. 2, we have modified our algorithm to handle the trade-off between mean waiting time and variance of waiting time. The ATWT value used as a selection criterion in Bucketing algorithm has been modified as in (2).

To observe the effect of different α values both on variance of waiting times and mean waiting time of Bucketing algorithm we conduct several experiments. In these experiments, α parameter of (2) is varied from 0.5 to 3.0, while using the other default parameter values given in Table 1. We observe that the trade-off between mean waiting time and variance of waiting time is evident. For higher values of α, the variance is improving, on the other hand, the mean waiting time of the algorithm is getting worse. The mean waiting time of the algorithm improves while α is increased from 0.5 to a certain value, which is 0.9 in our experiment. Then, for the values larger than this threshold, the mean waiting time begins to worsen again. This result is due to the fact that for the α values higher than 1, the modified ATWT heuristic in (2) attaches more importance to the waiting time of the first request than the total number of pending requests of a page. Therefore, the heuristic selects the pages similar to those selected with FCFS. On the other hand, when α is set to values lower than 1, the heuristic behaves in favor of the most requested pages like MRF. As a result, as the α value gets smaller or larger than 1, the experienced mean waiting time becomes more similar to that of one of the two algorithms.

We also conducted an experiment to compare the variance obtained with all scheduling algorithms. The results depicted in Fig. 3 show that FCFS has the lowest degree of variance due to the fact that in the worst case, the algorithm broadcasts any requested data item after broadcasting the whole set of data items in the database. That is, for the waiting time of a request, there is an upper bound which is determined by the database and page sizes. This upper bound also limits the variance of the waiting time. However, we can not claim this argument for the other algorithms.

The performance results obtained after modifying the computation of ATWT in our algorithm as in (2) are presented in Fig. 3. In this experiment, we set the α value to 2. For the high workloads, the modified ATWT has even better variance of waiting time than that of FCFS. However, as discussed above, the mean waiting time of the modified ATWT has become slightly worse (see Fig. 4). With a greater value for α (e.g., 3), the variance of waiting time can be further decreased; nonetheless, the the mean waiting time would become worse.

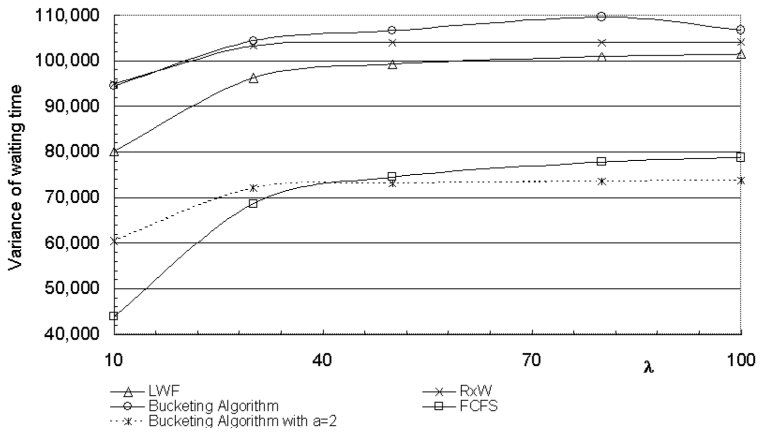


Fig. 3. Comparing Bucketing algorithm when α=2

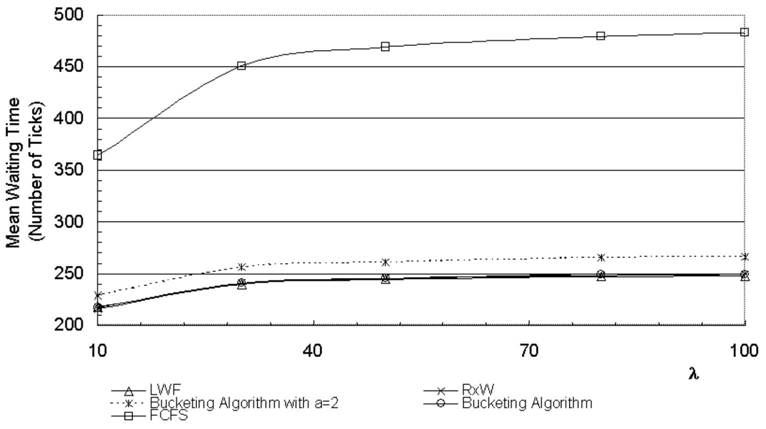


Fig. 4. Mean waiting time of Bucketing algorithm when α=2

4.5 Worst Waiting Time

The reason to investigate the worst waiting time is to check if a scheduling algorithm causes starvation of any request, which is an important property that should be avoided in interactive applications. Figure 5 displays the results for the longest waiting time experienced by any MU during the whole simulation time. For the default values of the simulation parameters presented in Table 1, FCFS has the lowest worst waiting time among all the algorithms. As discussed above, when FCFS is employed as the scheduling algorithm, any requested data item will be broadcast after the data items previously requested and the number of these data items is limited by the database size. In other words, the largest possible worst waiting time of a request is the time taken by broadcasting all the database items. However, for the other algorithms, it might be possible that a request waits while some of the data items are broadcast multiple times. Bucketing algorithm has considerably lower worst waiting time values compared to RxW. The results obtained with MRF are so much larger than those of the other algorithms that we do not include them in the figure.

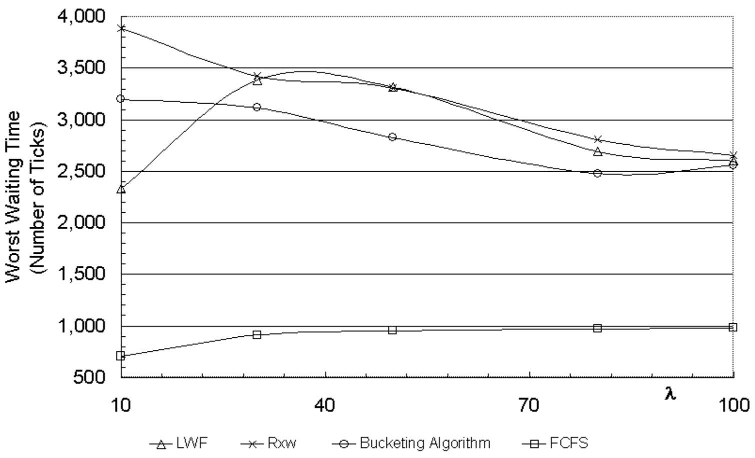


Fig. 5. Worst waiting time

4.6 Scheduling Decision Overhead

As discussed previously, a good scheduling algorithm should not have much scheduling decision overhead. In implementing the performance model, the decision overhead associated with each algorithm has not been considered. Since this overhead can not be accurately simulated for different types of algorithms, the time spent during the decision process has been ignored. For a comparative evaluation of decision overhead of the scheduling algorithms, we examined the number of requests scanned for selecting one of them to broadcast. If the number is large, the decision takes much more time and may become a bottleneck.

We compared the average number of data items scanned by three algorithms: LWF, RxW and Bucketing. FCFS is not included in this experiment. It just broadcasts the request that has arrived first, and does not need to compare any entry. On the other hand, its overall waiting time is so bad that it is not a competitive algorithm to be used.

As depicted in Fig. 6, LWF has the highest decision overhead while Bucketing algorithm has the lowest. The overhead of RxW is in between. Compared to other algorithms, Bucketing algorithm examines significantly fewer number of requests at each scheduling decision. For a request rate of 10 requests per tick,

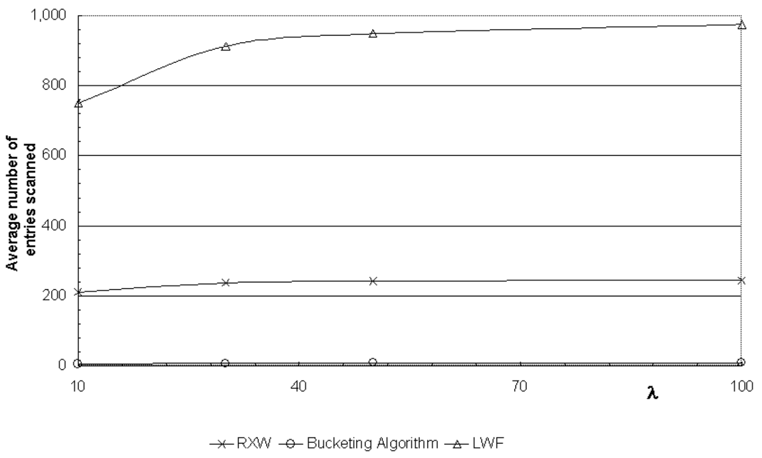


Fig. 6. Decision overhead

LWF compares 130 times more entries and RxW compares 36 times more entries than that of Bucketing algorithm[3].

4.7 Improving the Bucketing Algorithm

We have tried to improve the mean waiting time of our Bucketing algorithm and implemented the *depth* approach as presented in Sect. 3.2. There is a trade-off between the decision overhead and the mean waiting time in this approach. As we increase the search depth, we need to compare more entries of ATWT values, and we obtain lower mean waiting time. When we set the depth parameter to 50, the resulting mean waiting time of Bucketing algorithm is less than that of RxW as can be seen in Fig. 7.

5 Conclusion

In this paper, the problem we attack is the design of a broadcast scheduling algorithm which efficiently meets the demands of a mobile computing environment and mobile users. We have first proposed a new broadcast scheduling heuristic, ATWT, which is an approximate version of LWF heuristic [19]. Then, we have developed an algorithm called Bucketing algorithm to implement ATWT heuristic by using a bucketing scheme. Finally, we have conducted extensive simulation

[3] In [5], approximate versions of the RxW algorithm are proposed and they are shown to lead to less comparisons in deciding which data item to broadcast. However, these approximate versions have worse mean waiting time compared to RxW. The Bucketing algorithm, as discussed in Sect. 4.3, produces almost the same mean waiting times as RxW, while leading to less scheduling decision overhead.

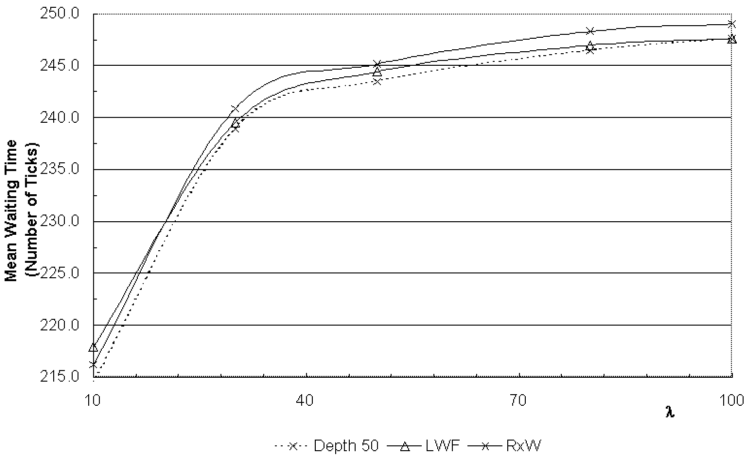


Fig. 7. Bucketing algorithm with depth=50 and the other competitive algorithms

experiments to evaluate the performance of our algorithm, and to compare the performance results against those of previously proposed scheduling algorithms.

Considering the performance results, the first remark to be done is that the most competitive algorithm to our algorithm is RxW [5]. The other algorithms, except LWF, do not produce good results with respect to the main performance criterion, the *overall mean waiting time.* Although, LWF has better performance than all the others, it has serious drawbacks which prevent its practical usage. In terms of the other performance metrics, i.e., *variance of waiting time* and *worst waiting time*, the performance of Bucketing algorithm is better, in general, than that of all other scheduling algorithms.

References

1. Acharya, S., Alonso, R., Franklin, M., Zdonik, S.: Broadcast disks: Data management for asymmetric communication environments. Proceedings of ACM SIGMOD (1995) 199–210
2. Acharya, S., Franklin, M., and Zdonik, S.: Balancing push and pull for data broadcast. Proceedings of ACM SIGMOD Conference. Tuscon, Arizona. (1997)
3. Acharya, S., M., Franklin, J., Zdonik, S.: Dissemination-based data delivery using broadcast disks. IEEE Personal Communications. **2(6)** (December 1995) 50–60
4. Aksoy, D., Franklin, M.: Scheduling for large-scale on-demand data broadcasting. Proceedings of the IEEE INFOCOM Conference. (1998) 651–659
5. Aksoy, D., Franklin, M.: Rxw: A scheduling approach for large-scale on-demand data broadcast. ACM/IEEE Transactions on Networking. **7** (1999) 846–860
6. Ammar, M., H., Wong, J., W.: The design of teletext broadcast cycles. Performance Evaluation. **5** (November 1985) 235–242
7. Ammar, M., H., Wong, J., W.: On the optimality of cyclic transmission in teletext systems. IEEE Transactions on Communications. **35** (January 1987) 68–73

8. Barbara, D.: Mobile computing and database: A survey. IEEE Transactions on Knowledge and Data Engineering. **11** (January-February 1999) 108–117
9. Barbara, D., Imielinski, T.: Sleepers and workaholics: Caching strategies in mobile environment. ACM SIGMOD RECORD. **23** (May 1994)
10. Hameed, S., Vaidya, N.,H.: Log-time algorithms for scheduling single and multiple channel data broadcast. Proceedings of the Third Annual ACM/IEEE International Conference on Mobile Computing and Networking. Budapest, (September 1997) 90–99
11. Hameed, S., Vaidya, N.,H.: Efficient algorithms for scheduling data broadcast. Wireless Networks. **5** (1999) 183–193
12. Imielinski, T., Badrinath, B., R.: Mobile wireless computing: Challenges in data management. Communications of the ACM (October 1994) 19–27
13. Jiang, S., Vaidya, N., H.: Response time in data broadcast systems: Mean, variance and trade-off. Proceedings of International Workshop on Satellite-based Information Services (WOSBIS). (October 1998)
14. Jiang, S., Vaidya, N., H.: Scheduling data broadcasting to “impatient” users. Proceedings of the ACM International Workshop on Data Engineering for Wireless and Mobile Access. Seattle, WA USA, (August 1999) 52–59
15. Karakaya, M., Ulusoy, Ö.: An efficient broadcast scheduling algorithm for pull-based mobile environments. Submitted to the IEEE/ACM Transactions on Networking.
16. Schwetman, H.: Csim18 the simulation engine. In J., Charnes, D., Morrice, D., Brunner, J., Swain, editors. Proceedings of the 1996 Winter Simulation Conference. San Diego, CA, (1996) 517–521
17. Vaidya, N., H., Hameed, S.: Scheduling data broadcast in asymmetric communication environments. Wireless Networks. **5** (1999) 171–182
18. Vaidya, N., H., Jiang, S.: Data broadcast in asymmetric wireless environments. Proceedings of First International Workshop on Satellite-based Information Services (WOSBIS). NY, (November 1996)
19. Wong, J., W.: Broadcast delivery. Proceedings of The IEEE. **76** (December 1988) 1566–1577
20. Zipf, G., K.: Relative frequency as a determinant of phonetic change. **XL**. Reprinted from the Harvard Studies in Classical Philiology. (1929)

An Architecture for Workflows Interoperability Supporting Electronic Commerce*

Vlad Ingar Wietrzyk[1], Makoto Takizawa[2], and Vijay Khandelwal[1]

[1] School of Computing, University of Western Sydney
v.wietrzyk@uws.edu.au, Australia
[2] Department of Computures and Systems Engineering
Tokyo Denki University, Japan

Abstract. *D*istributed object architectures (DCOM, CORBA, Java RMI) coupled with Internet and Intranet technology have a great impact on process–centered environments and distributed workflow management. Management of the workflow operations of each organisation in the interworkflow must be done in a distributed environment.
An electronic commerce (EC) process is a business process and defining it as a workflow provides all the advantages that come with this technology. This paper describes the design of a model as well as an architecture to provide support for distributed advanced workflow transactions. The componentwise architecture of the system makes it possible to incorporate the functionality and thus the complexity only when it is actually needed. We discuss the application of transaction concepts to activities that involve integrated execution of multiple tasks over different processes. This kind of application is described as transactional workflows.

1 Introduction

Distributed workflow execution across functional domains is necessary, but distribution transparency is currently impossible because, different types of Workflow-Management-Systems (WFMSs) implement different WFMS metamodels.

For smooth cooperation among organizations, the most important point is how the hierarchy of the cooperation process among organizations realizes. We can call the interoperation of workflows iterworkflow and the total support technologies, which are necessary for its realization, interworkflow management mechanism. Interworkflow is anticipated as a supporting mechanism for Business-to-Business Electronic Commerce.

One possible way to enable distributed workflow execution is to build a workflow-management infrastructure integrating different and heterogeneous WFMSs. Users would have access to total funcionality because they access the workflow-management underlying infrastructure, not individual WFMSs. The resulting architecture is general and can accommodate as many WFMSs as required.

* Research supported by the **UWS, Versant Technology Corporation** and **Intel Corporation**

Transaction concepts have begun to be applied to support applications or activities that involve multiple tasks of possibly different types - including, but not limited to transactions, and executed over different types of entities - including DBMSs. The architect of such applications may specify inter-task dependencies to define task coordination requirements, and additional requirements for isolation, and failure atomicity of the application. Generally we will refer to such applications as multi-system transactional workflows.

Workflow applications are long-duration applications since the duration of a workflow can range from a few hours to a few months.

To summarize, the new aspects of our approach to distributed workflow database management systems supporting e-commerce include the following researh contributions. The novel approach to the notions of correctness for transaction processing. Usually proposed the definition of correctness for multiuser databases are not necessarily suitable when these databases are parts of a multilevel secure workflow systems. We believe, that the best approach will depend upon the characteristics of the multilevel workflow database and the applications.

Designing the high scalability and secure architecture enabling efficient e-commerce operations which allows an arbitrary combination of transaction support systems and workflow management systems to which the locations are irrelevant is an objective of our ongoing research.

1.1 Outline of the Paper

We have planned the presentation of the current research as follows. We first present a brief introduction to work on workflow transaction models and discuss extended – relaxed approach to handle workflow transactions in section 2. Section 3 covers related aspects of workflow distribution and heterogeneity. A number of relaxed transaction models in workflow contexts that have been defined recently permitting a controlled relaxation of the transaction isolation and atomicity to better match the requirements of various workflow applications are discussed in section 4. In section 5 we consider two aspects of workflow interoperability: the interoperability protocol between independent WFMSs and the ability to model the interoperability in a workflow process definition tool. Section 6 concludes the paper with a summary and a short discussion of future research.

2 Related Work

Some known examples of extended transaction models include nested and multilevel transactions. A number of relaxed transaction models have been defined in the last several years that permit a controlled relaxation of the transaction isolation and atomicity to better match the requirements of various advanced applications. Some examples of extended – relaxed transaction models are reported in [1, 2].

In the WIDE project [3], a workflow is supported at two transaction levels: global and local. At the global level, the SAGA - based model offers relaxed atomicity through compensation and relaxed isolation by limiting the isolation to the SAGA steps. At this level of granularity the workflow activities are defined and therefore the grouped workflow activities follow the strict ACID properties. However, the flexibility in assigning transaction properties to workflow activities is limited to the extended SAGA or nested transaction model.

Support for long-duration applications has been independently extended by practitioners and researchers focusing on workflow systems and transaction systems. Extended transaction systems structure a large transaction into sub-transactions and execute them with additional constraints on the individual sub-transactions. Some researchers in workflow systems have proposed the notion of transactional workflow [4]. In transactional workflow environment, additional correctness requirements can be specified on top of traditional workflow specifications.

The Workflow Management Coalition has specified a standard interface to facilitate the interoperability between different WFMSs [5]. However, they do not address transactional issues with the exception of writing an audit log.

The transaction model used in the Exotica project [6] is based on the SAGA model, but relies on statically computed compensation patterns. As a result, its functionality is limited compared to the work presented in this research paper.

Finally, most commercial products are designed around a centralized database. This database and the workflow engine attached to it — in most cases there is a single workflow engine are a single point of failure which quickly become a bottleneck and are not capable of providing a sufficient degree of fault tolerance.

To summarize, databases and workflow management systems are complementary tools within the corporate computing resources. Databases address the problem of storing and access data efficiently. Workflow management systems address the problem of monitoring and coordinating the flow of control among heterogeneous activities.

3 Workflow Distribution and Heterogeneity

Management of the workflow operations of each organization in the interworkflow must be done in a distributed environment. Because the interworkflow defines the process of linking among different workflows, naturally the operations management takes place across different workflows. The interworkflow for international procurement, for example, includes the procurement workflow and also the ordering/delivery management workflow of each supplier company; and these workflows are likely to be distributed across the procuring company's system and the systems of each of the supplier companies.

Workflow distribution introduces additional level of requirements. Because distributed workflow execution across heterogeneous WFMSs is currently not possible in a transparent way, we must to consider the problem of workflow funcionality isolation.

To efficiently define a work breakdown structure a functional decomposition by means of subworkflows is required. This in turn requires version and variant management, because each reused-workflow-definition alteration might lead to a new variant or configuration of the reusing workflow definition, if it necessitates to stay related to the old version. Dynamic changes of running workflows require a workflow's functional decomposition to change. This might also involve replacing elementary workflow tasks with other composite workflow definitions. Workflow distribution is called homogeneous if the associated WFMSs are of the same type, heterogeneous otherwise. If the involved WFMSs are heterogeneous, this might nessecitate data, (object)-type translation on the side of the requesting or receiving WFMS. A workflow is distributed when at least two of its objects reside in two different WFMS installations. This is relevant to workflow definitions as well as workflow instances. An often-cited situation is subworkflow distribution, where subworkflows are subject to excution on remote WFMSs. Some variants are possible, such as executing a subworkflow synchronously or asynchronously to the invoking workfow. One of the typical variant involves executing some part of a workflow on one WFMS, and continuing on another (see Figure 1). If the associated WFMSs, do not know about each other, it's

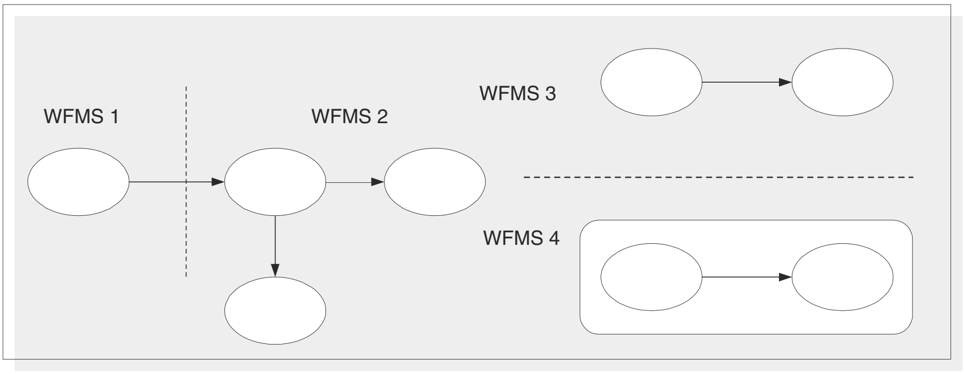


Fig. 1. Workflows Division across different WFMSs

indirect distribution. In this case, the WFMSs do not implement distribution natively, and system designer must attach distribution functionality to the associated WFMSs. A recognised way is to establish communication buffers between the WFMSs, such as a database or persistent file stores. Figure 2 shows an example workflow definition with one distribution task. The distribution task invokes an application for buffer communication. Typically, workflow types can be distributed, too.

3.1 An Architecture for Multilevel Secure Workflow Interoperability

Global information management strategies based on a sound distributed architecture are the foundation for effective distribution of complex applications that are needed to support ever changing operational conditions across security boundaries. What we need is a new Multilevel Secure Workflow (MLS) distributed computing paradigm that can assist users at different locations and at different security levels to cooperate.

A user can initiate a distributed workflow transaction at any site. If access to objects stored at remote sites is required, the distributed workflow transaction initiates a subtransaction at the remote site. To guarantee correct execution of distributed workflow transactions, each site in the distributed workflow database is under the operation of a concurrency control protocol and an atomic commit protocol.

We present the fully distributed architecture for implementing a Workflow Management System (WFMS). An MLS workflow distributed database consists of a set $\mathcal{N}$ of sites, where each site $N \in \mathcal{N}$ is an MLS database. The sites in the workflow system are interconnected via communication links over which they can communicate. The WFMS architecture operates on top of a Common Object Request Broker Architecture (CORBA) implementation. A CORBA's Interface Definition Language (IDL) is used to provide a means of specifying workflows. Also we assume that communication links are secure — possibly using encryption. This distributed workflow transaction processing model describes mainly those components necessary for the distribution of a transaction on different domains.

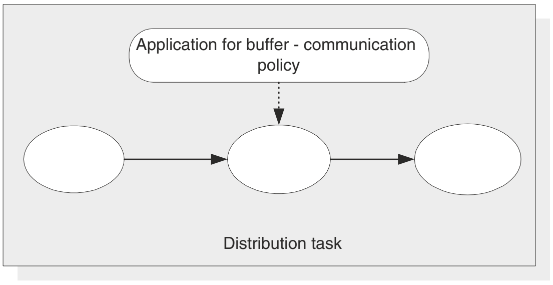


Fig. 2. The Distribution Task Invokes an Application for Buffer Communication

Domain is a unit of autonomy that owns a collection of flow procedures and their instances. In practical terms, a domain might define the scope of a department or division in an organization. Therefore, flows are grouped by domains, and each domain also manages a set of flow procedures installed in the domain. A domain is not defined or limited by networks, processors, or peripherals. The manager of resources can, however, be designed in any fashion, they are exclusively responsible for the ACID properties on their data records. Solely the interface to the components of the distributed workflow model must exist.

If a transaction should be dstributed on several domains — a global transaction, in every domain there must exist the following components, (see Figure 3).

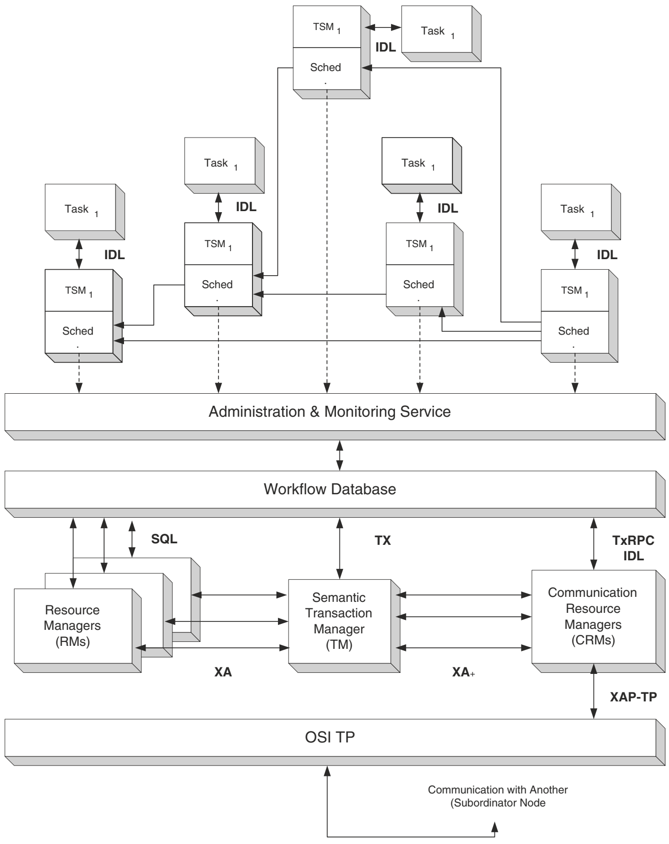

TSM 1
Task 1
IDL
Sched
Administration & Monitoring Service
Workflow Database
SQL
TX
TxRPC
IDL
Resource Managers (RMs)
Semantic Transaction Manager (TM)
Communication Resource Managers (CRMs)
XA
XA+
XAP-TP
OSI TP
Communication with Another (Subordinator Node

Fig. 3. Distributed Workflow Architecture

– TM - Transaction–Manager. The transaction manager plays the role of the coordinator in the respective domain. If a transaction is initiated in this domain, the TM assigns a globally unique identifier for it. The TM monitors all actions from applications and resource managers in its domain. In every domain involved in the distributed workflow transaction environment there exists exactly one TM.
– CRM - Communication–Resource–Manager. Multiple applications in the same domain talk with each other via the CRM. This module is used by applications but also other management components for inter-domain communication. CRM is the most important module with respect to the transactional

support for distributed workflow executions. Our model specifies the T*RPC as a communication model, which supports a remote procedure call (RPC) in the transactional environment.

- RM - Resource–Manager. An accountable performer of work. A resource can be a person, a computer process, or machine that plays a role in the workflow system. This module controls the access to one or more resources like files, printers or databases. The RM is responsible for the ACID properties on its data records. A resource has a name and various attributes defining its characteristics. Typical examples of these attributes are job code, skill set, organization unit, and availability.
- AMS - Administration–Monitoring–Service. The monitoring manager is used to control the workflow execution. In our approach, there is no centralized scheduler. In the figue, each Task Manager - designated as TSM, is equipped with a conditional fragment of code which determines if and when a given task is due to start execution. The scheduler communicates with task managers using CORBA's asynchronous Interface Definition Language(IDL) interfaces. Task managers communicate with tasks using synchronous IDL interfaces as well. AMS module is also responsible for the coordination of the different sites in case of an abort that involves multiple sites. Individual task managers communicate to monitoring manager their internal states, as well as data object references - for possible recovery.

The distributed architecture suits the inherent distributional character of workflow adequately in a natural way.

This approach also eliminates the bottleneck of task managers having to communicate with a remote centralized scheduler during the execution of the workflow. This architecture also posseses high resiliency to failures — if any one node crashes, only a part of the workflow is affected.

4 Relaxed Transaction Models in Workflow Contexts

A number of relaxed transaction models have been defined recently that permit a controlled relaxation of the tranaction isolation and atomicity to better match the requirements of various workflow applications. Usually, we will refer to such applications as multi-system transactional workflow. This area has been also influenced by the concept of long running activities.

The intenton is to merge advanced transaction technology and workflow management systems to support business processes with well-defined failure semantics and recovery features. Our work is based on an interpretation of the workflow operations from the database point of view.

As has been pointed out in [7], WFMSs lack the ability to ensure the correctness and reliability of the workflow execution in the presence of concurrency and failures. When the task is a transaction executed by a DBMS providing a full range of transaction management functions, we can take advantage of its concurrency control, commitment, recovery and access permit facilities. However, when the task is executed by an application system, we must understand the application system semantics that affects its transactional behavior.

4.1 Transactional Workflows

Support for workflow applications has been addressed by researchers focusing on workflow systems and transaction systems. Extended transaction systems structure a large transaction into sub-transactions and execute them with additional precedence requirements between start – commit – abort of the individual subtransactions. Our approach falls in the category of transactional workflows [4] where additional correctness requirements can be specified on top of traditional workflows specifications. These requirements specify additional constraints on workflow execution schedules. Workflow management systems coordinate the execution of applications distributed over networks. The need for coordinated execution of workflow steps arises from application as well as data consistency requirements. Flexible transactions work in the context of heterogeneous distributed multidatabase workflow environments [8]. In such workflow environments, each database acts independently from the others. Because a local database can unilaterally abort a transaction, it is not possible to enforce the commit semantics of global transactions. Therefore, flexible transaction were designed to address this problem.

4.2 The Functionality of Flexible Transactions in Workflow Systems

A flexible transaction is specified by providing: the precondition of the global transaction, a set of subtransactions, the externally visible states of each subtransaction and the possible transitions among these externally visible states, preconditions and postconditions for the possible transitions of each subtransaction, and the postcondition of the global transaction.

The traditional transactions are usually characterized by the atomicity, consistensy, isolation and durability requirements, called the ACID properties of transactions. To better support workflow operational environment, the flexible transaction model relaxed the isolation and atomicity properties. This approach is the direct result of our believe, that tying a workflow system to a particular transaction model, will result in major restrictions that will limit its applicaility and usefulness as a workflow tool.

4.3 A Formal Model of Flexible Transactions

From a user's point of view, a transaction is a sequence of actions performed on data items in a database. Flexible transaction model proposed for the distributed workflow environment will increase the failure resiliency of global transactions by allowing alternate subtransactions to be executed when a local database fails or a subtransaction aborts. The approach supports the concept of varied transactions allowing compensatable and noncompenstable subtransactions to coexist within a single global transaction. This transactional environment allows a global transaction to have a weaker (relaxed) form of atomicity, termed semi-atomicity, while still maintaining its correct execution in the workflow. In a workflow multidatabase environment, a local transaction is a set of subtransactios, where each subtransaction is a transaction accessing the data items at a single local site. The concurrency control of global transactions require, that each global transaction has at most one subtransaction at each local site [9]. Following [8, 10], the definition of flexible transactions takes the form of a high-level specification. The flexible transaction model supports flexible execution control flow by specifying two kinds of dependencies among the subtransactions of a global transaction:

– Execution ordering dependencies between two subtransactions.
– Alternative dependencies between two subsets of subtransactions.

In what follows, we shall formally describe the flexible execution control in the flexible transaction model.

Let $\Omega = \{t_1, t_2, \ldots, t_n\}$ be a collection of subtransactions and $\Pi(\Omega)$ the collection of all subsets of Ω. Let t_i, $tj \in \Omega$ and T_i, $T_j \in \Pi(\Omega)$. Two types of control flow relations are defined on the subsets of Ω and on $\Pi(\Omega)$, namely:

– precedence $t_i \prec t_j$ if t_i precedes t_j $(i \neq j)$;
– preferece $T_i \rhd T_j$ if t_i is preferred to T_j $(i \neq j)$. If $T_i \rhd T_j$, we also declare that T_j is an alternative to T_i.

Both of the above relations, precedence and preference are irreflexive and transitive or more formally, for each $t_i \in \Omega$, $\neg(t_i \prec t_i)$; and for each $T_i \in \Pi(\Omega)$, $\neg(T_i \rhd T_j)$. If $t_i \prec t_j$ and $t_j \prec t_k$, then $t_i \prec t_k$; if $T_i \rhd T_j$ and $T_j \rhd T_k$, then $T_i \rhd T_k$.

From he above definitions, we can see than, the precedence relations determines the correct parallel and sequential execution ordering dependencies among the subtransactions, while the preferece relation determines the priority dependencies among alternate sets of subtransactions for selecting in completing the execution of Ω.

Now a flexible transaction can be defined as follows:

Definition 1. Flexible transaction *A flexible transaction Ω is a set of related subtransactions on which the precedence $(\prec)$ and preference $(\rhd)$ relations are defined.*

The semantics of the precedence relation refers to the execution order of subtransactions. For example, $t_1 \prec t_2$ may imply that t_2 cannot start before t_1 finishes or that t_2 cannot finish before t_1 finishes. By the same token, the preference relation defines alternative choices and their priority. For example, $\{t_i\} \rhd \{t_j, t_k\}$ may imply that t_j and t_k must abort when t_i commits or that t_j and t_k should not be executed if t_i commits. In this environment, $\{t_i\}$ is of higher priority than $\{t_j, t_k\}$ to be chosen for execution.

We consider that a workflow database state is consistent if it preserves workflow database integrity constraints. As it is the case for traditional transactions, the execution of a flexible transaction as a single unit should map one consistent multidatabase workflow state to another. We designate relation $(T_i, \prec_i)$ as a partial order of subtransactions. $(T_i, \prec_i)$ is a representative partial order, if the execution of subtransactions in T_i represents the execution of the entire flexible transaction Ω. From the above it is clear that, if $(T_i, \prec_i)$ is a representative partial order, then there are no subsets T_{i1} and T_{i2} of T_i such that $T_{i1} \rhd T_{i2}$. Because each global transaction has at most one subtransaction at a local site, each representative partial order of a flexible transaction must have at most one subtransaction at a local site. In our workflow execution environment, for flexible transactions, the above definition of consistency requires that the execution of subtransactions in each representative partial order must map one consistent workflow multidatabase state to another.

4.4 Scheduling of Flexible Transactions

Since the flexible transaction model was proposed, much research has been devoted to its application. The availability of visible prepare–to–commit states in local database systems is the basic assumption underlying this work. In such an operational environment, the preservation of the semi-atomicity of flexible transactions is relatively easy. As we mentioned in the previous subsection, failures of subtransactions in a flexible transaction are tolerated by taking advantage of the fact that a given function can frequently be accomplished by more than one database system. Also, time used in conjunction with subtransaction and global transaction can be exploited in transaction scheduling.

A schedulable subtransaction may be submitted for execution to the transaction module. The scheduler first has to check for satisfaction of the preconditions for execution of each subtransaction — it determines whether a subtransaction is schedulable. This entails the specification of the execution dependency among the subtransactions of a global transaction. Execution dependency [4], is a relationship among subtransactions of a global transaction which determines the legal execution order of the subtransactions. To support the specification of the execution dependency, we define a transaction execution state as follows:

Definition 2. *The transaction execution state x for a global transaction T with m subtransactions, is an $m-tuple$ $(x_1, x_2, \ldots, x_m)$ where:*

$$x_i = \begin{cases} E & \textit{if } t_i \textit{ is currently being executed;} \\ N & \textit{if subtransaction } t_i \textit{ has not been} \\ & \textit{submitted for execution;} \\ S & \textit{if } t_i \textit{ has successfully completed;} \\ F & \textit{if } t_i \textit{ has failed or completed without} \\ & \textit{achieving its objective;} \end{cases}$$

Under normal operational circumstnces transaction execution state is used to keep track of the execution of the workflow subtransactions. It is also used to determine if a global workflow transaction has achieved its objectives. When a subtransaction t_i complete the corresponding execution state, x_i is set to S if the subtransaction has achieved its objective, and to F, therwise. At a certain point of execution, the objectives of the global workflow transaction may be achieved. At that point, the global transaction is considered to be successfully completed and can be committed.

A number of approaches can be used to assure global serializability which constitutes a satisfactory correctness criterion for concurrent execution of multidatabase workflow transactions, if there is a lack of additional information about their semantics. The objective of concurrency control is to assure that the serialization order of multidatabase workflow transactions should be the same, at all sites they execute. It was shown in [8, 11], that the above condition is sufficient to assure global serializability. However, in our workflow operational environment this requirement can be relaxed to require that the relative serialization order of Workflow Transactions should be the same only at those nodes where they conflict. This would lead to a weaker notion of serializability; called WT-serializability, which will be used as our correctness criterion for concurrent execution of Workflow Transactions. We define conflict among workflow transactions if they execute at the same (local) site, and they are not commutative. The conflict relation is transitive, and therefore determines a set of equivalence classes, which can be named as conflict classes. In our workflow environment they are used to determine the granularity of locking. In order to define workflow transaction serializability; WT-serializability, let us consider two workflow flexible transactions WT_α and WT_β, and conflict classes, i and j. A global schedule is WT-serializable if for any subtransactions ST_i^α and $ST_j^\alpha \in WT_\alpha$, and ST_i^β and $ST_j^\beta \in WT_\beta$ such that conflict $(ST_i^\alpha, ST_i^\beta)$ and conflict $(ST_j^\alpha, ST_j^\beta)$, $ST_i^\alpha \prec ST_i^\beta \Rightarrow ST_j^\alpha \prec ST_j^\beta$, at all sites they conflict. In our workflow environment the $\prec$ relationship is defined in terms of local serializability. WT-serializability establishes a partial order among all workflow flexible transactions. The submission order at each system, can be used to determine the execution and, consequently, the serialization order at each site. Therefore, the concurrency control mechanism of the local system will assure that the transactions that are submitted to the local system will be executed correctly with respect to the local concurrency control. As a result, the lock held by a subtransation can be released as soon as the subtransaction completes its submission phase. Therefore, we will have several transactions that are executing concurrently at each local site.

5 Interworkflow Management Mechanism

Composing an MLS workflow from multiple single-level workflows is the only practical way to construct a high-assurance MLS WFMS today. In this particular approach, also the multilevel security of our MLS workflow does not depend on a single-level WFMS, but rather on the underlying MLS distributed architecture. On our platform, an MLS workflow becomes multiple single-level workflows that will be run on the MLS architecture. These independent workflows should work together to achieve the intended operation. Therefore, workflow interoperability is an important issue.

We should consider two aspects of workflow interoperability:

- The interoperability protocol between independent WFMSs.
- The ability to model the interoperability in a workflow process definition tool.

The first issue can be defined and handled by a standards body such as OMG. However, the second aspect should be handled by each WFMS. OMG [12] introduces two models of interoperability. They are nested sub-process and chained processes as shown in Figures 4 and 5.

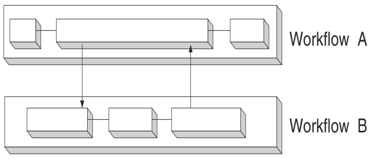


Fig. 4. Nested sub-process

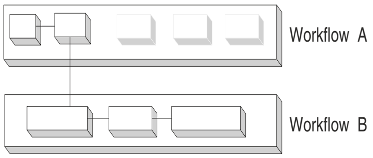


Fig. 5. Chained processes

In nested sub-process workflow design, a task in workflow A may invoke workflow B in a role as the actor of a particular task and then wait for it to complete. Therefore, the task in workflow A is a requester, and the task that is realized by the sub-process can act as the coordination point for interaction of the two workflows.

In chained workflow design, a task may invoke another, then carry on with its own business processing logic. In this case, the task associated with the sub-process would be another entity that is stimulated by the results of the sub-process. The workflows will terminate independently of each other. The nested sub-process and chained processes models provide powerful mechanisms for workflow interoperability.

However, we would like to extend the above two models to support a richer interoperability model: cooperative processes. Very offen there is a need to consider two independent autonomous workflows that need to cooperate across two different organizations. Let's assume that organization A is in control of workflow A and Organization B is in charge of workflow B. Than, tasks in workflow A and workflow B can communicate and synchronize with each other as shown in Figure 6. Generally, two or more cooperating workflows may have indepen-

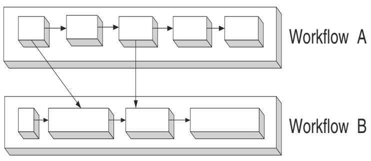


Fig. 6. Cooperative processes

dent starting and ending points. Let's state a minimal set of information that is required for communication and synchronization among tasks in cooperating autonomous workflows. These should include:

– protocol to establish the location and invocation method of tasks
– protocol to coordinate receiving of replies

In order to ensure the proper generation of the runtime code, the above specification has to appear in the workflow design. Such environment will guarantee, that execution will proceed according to the workflow control protocol.

6 Conclusion

The impetus for our current research is the need to provide an adequate framework for interworkflow as it is anticipated that it will be a supporting mechanism for Business-to-Business Electronic Commerce. We proposed the following auxiliery functions as a necessary mechanism for implementing operations management of workflows in a disributed environment to support Electronic Commerce:

– A mechanism for controlling execution of workflows designed on other workflow management systems.
– A mechanism for moitoring the status of workflows running on other workflow management systems.
– The global-business characteristic of Electronic Commerce workflows requires a high-throughput workflow execution engine. Therefore, to ensure this kind of scalability, load distribution across multiple workflow servers is necessary.
– The necessity of efficient replication of workflow servers. This will ensure that the market position of the merchant will not be weakend due to frequent failures and unavailability of Electronic Commerce servers.

The notions of correctness for transaction processing that are usually proposed for multiuser databases are not necessarily suitable when these databases are parts of a multilevel secure workflow systems. We believe, that the best approach will depend upon the characteristics of the multilevel workflow database and the applications. It is incumbent upon those who develop multilevel database systems to ensure that the user's needs and expectations are met to avoid misunderstandings about the system's functionality.

The insight developed in the current research serves as the basis for the next step, which is to build application specific software architecture that encode business logic for coordinating widely distributed manual and automated tasks in achieving enterprise level objectives.

We choose to develop framework for distributed workflow architecture as the foundation of workflows interoperability supporting e-commerce. There are some aspects of workflows interoperability supporting e-commerce which needs to be addressed to make the distributed workflow architecture more useful. In our future work we will focus on providing support for adaptive workflow so that workflow can be altered at runtime.

References

1. V. Wietrzyk, Mehmet A. Orgun. A Foundation for High Performance Object Database Systems. In *Databases for the Millennium 2000 in proceedings of the 9th International Conference on Management of Data*, Hyderabad, December, 1998.
2. A. Elmagarmid. Transaction Models for Advanced Database Applications. *Morgan-Kaufmann*, February 1992.
3. G. Grefen, B. Pernici, G. Sanchez. Database Support for Workflow Mnagement - The WIDE Project. In *Kluwer Academic Publishers*, August 1999.
4. M.Rusinkiewicz and A. Sheth. On transactional Workflows. *Bulletin of the Technical Committee on Data Engineering.* (June 1993).
5. The Workflow Management Coalition Interoperability Abstract Specification. *The Workflow Management Coalition* , June 1996
6. G. Alonso and D. Agrawal Advanced transaction Models in Workflow Contexts. *Procs. Int. Conf. on data Engineering.* 1996.
7. D. Georgakopoulos, M. Hornick, and A. Sheth. An Overview of Workflow Management: From Process Modeling to Workflow Automation Infrastructure. *Distributed and Parallel Databases*, 3(2):119-153, April 1999.
8. A. K. Elmagarmid, Y. Leu, W. Litwin, and M. E. Rusinkiewicz. A Multidatabase Model for Interbase. In *Proc. of the 16th VLDB Conference*, August 1990.
9. V. Gligor and R. Popescu-Zeletin. Transaction Maagement in Distributed Heterogeneous Database Management Systems. In *Information Systems*, 11(4), 1986.
10. A. Zhang, M. Nodine, B. Bhargava, O. Bukhres Scheduling with Compensation in Multidatabase Systems. In *CSD-TR-93-063*, 11(4), October 1993.
11. M. Ansari, M. Rusinkiewicz, L. Ness, A. Sheth Executing Multidatabase Systems. In *TM-TSV-019450*, July 1991.
12. Object Management Group. The Object Request Broker. *Architecture and Specification*, 2000.

Object and Log Management in Temporal Log-Only Object Database Systems

Kjetil Nørvåg[⋆]

Department of Computer and Information Science
Norwegian University of Science and Technology
7491 Trondheim, Norway
Kjetil.Norvag@idi.ntnu.no

Abstract. We have previously studied the possible performance gain from using the log-only approach to realize temporal object database systems. Although the log-only approach in its basic form is relatively straightforward, it is not trivial to support features such as steal/no-force buffer management, fuzzy checkpointing, and fast commit. In this paper, we describe in detail algorithms and strategies for object and log management that make support for these features possible.

1 Introduction

Many emerging application areas for database systems demand features and performance difficult to support by todays systems. Common for many of these, is the need for management of complex objects, support for temporal objects, and support for querying changes (for example, updates since a particular time T). Examples of such application areas are XML/Web databases, geographical information systems and PACS (Picture Archiving and Communications Systems for use in health care).

An interesting solution to support the desired features, is *log-only object database systems*. The log is written contiguously to the disk, in a no-overwrite way, in large blocks. This is done by writing many objects and index entries, possibly from many transactions, in one write operation. This gives good write performance, but possibly at the expense of read performance. This contrasts to most current object database systems (ODBs), where data is updated in-place. In order to support recovery and increase performance, write-ahead logging can be used. This logging defers the in-place update, but sooner or later, the update has to be done. This often results in the writing of lots of small objects, creating a write bottleneck. This bottleneck can be avoided by using the log-only approach.

The log-only approach has many features that makes it interesting. Two examples are fast recovery and ability to benefit more from using RAID technology than traditional systems. A third feature, which is the motivation for our research, is the fact that keeping previous versions of objects comes almost for free when using a log-only approach. This is particularly interesting for transaction-time object database systems (TODB), where object updates do not make previous versions unaccessible. On the contrary, previous

[⋆] The author supported in part by NFR (Research Council of Norway) and ERCIM (European Research Consortium for Informatics and Mathematics).

versions of objects can still be accessed and queried, and a system maintained timestamp (commit time of the transaction that created a version of an object) is associated with every object version. In a traditional system with in-place updating, keeping old versions of objects, which is required in a transaction-time temporal database system, usually means that the previous version has to be copied to a new place before update. This doubles the write cost. With the log-only approach, this is not necessary.

Previous log-only object database systems have been page server based. While this works well in many contexts, it is not ideal. By operating on page granularity you get many of the disadvantages of traditional pager servers. For example, if clustering is bad, and only a small part of a page has been updated, it is still necessary to write back the whole page. With bad clustering, main memory buffer utilization will be bad as well. A page based log-only ODB also makes transaction management difficult. To avoid page level locking, you essentially need to have 1) a separate log anyway, or 2) use ad-hoc techniques to solve the problem. Both solutions are likely to hurt performance and increase complexity, and have together with performance analysis of log-only TODBs [3] convinced us that an object based log-only TODB is the way to go.

Although the log-only approach in its basic form, and using page granularity, is relatively straightforward, more effort is needed in order to achieve performance and high concurrency. In this paper, we describe the Vagabond approach for efficient storage and management of temporal objects. With the Vagabond approach, some of the problems in related designs are avoided, and steal/no-force buffer management, fuzzy checkpointing and fast commit is supported.

The organization of the rest of the paper is as follows. In Sect. 2 we give an overview of related work. In Sect. 3 we describe the Vagabond log-only approach. In Sect. 4 we describe in detail the algorithms for the most important operations in the Vagabond approach. Finally, in Sect. 5, we conclude the paper.

2 Related Work

No-overwrite strategies have been used in shadow-paging recovery strategies earlier, e.g., in System R [1], but with the limited buffer size at that time, the performance was not satisfactory. POSTGRES [9] also employed a no-overwrite strategy, but had also its performance problems for several reasons, the most important being the buffer force strategy used.

The Vagabond approach is based on the same philosophy as log-structured file systems (LFS), which was introduced by Rosenblum and Ousterhout [7]. LFS has been used as the basis for object stores, including the Texas persistent store [8]. Those object stores are page based, i.e., when an object has been modified, the whole page it resides on has to be written back. To our knowledge, there has been no publications on log-only ODBs operating on *object* granularity as in the Vagabond approach.

A discussion of some topics related to TODBs can be found in [4]. More details about the Vagabond approach can be found in the author's doctoral thesis [6].

3 The Vagabond Approach

In this section we give an overview of our approach to log-only TODBs. We give an overview of log management, storage objects, and indexing.

3.1 Introduction to Log-Only Log Management

With the log-only approach, already written data is never modified, new versions of the objects are just appended to the log. Logically, the log is an infinite length resource, but the physical disk size is, of course, not infinite. This problem is solved by dividing the disk into large, equal sized, physical segments. When one segment is full, writing is continued in the next available segment. As data is vacuumed, deleted or migrated to tertiary storage, old segments can be reused. Dead data, in a TODB most often old index nodes, will leave behind partially filled segments, the data in these near empty segments can be collected and moved to a new segment. This process, which is called *cleaning*, makes the old segments available for reuse. By combining cleaning with reclustering, we can get well clustered segments.

A segment can be in one of three states. A segment starts in a *clean state*, i.e., it contains no data. The segment currently being written to, is called the *current* segment. When the segment is full, we start writing into a new segment. The new segment now goes from the *clean* state, to *current*. The previous segment is now *dirty*, it contains valid data (note that dirty in this context has nothing to do with main-memory state versus disk state, as the term is most frequently used). Information about the status of the segments is kept in the *segment status table* (SST), which is kept in main memory during normal operation.

At regular times, a checkpoint operation is performed. In the checkpoint operation, we write enough information to the log to make the current position in the log a consistent starting point for recovery. The checkpoint information is stored in *checkpoint blocks*, which are stored in fixed positions in the log.

Each new version of an object is written to a new place, making the use of logical object identifiers (OIDs) necessary. When using logical OIDs, an OID index (OIDX) is needed to do the mapping from logical OID to physical location when retrieving an object. The index entries in the OIDX, the *object descriptors* (OD), contains the physical address for an object and the commit timestamp. Note, however, that the ODs that are written together with the objects to the log contain transaction identifiers (TIDs), and the mapping from TID to commit time is written to the log as part of the commit operation. The ODs in the OIDX from committed transactions however, contains commit timestamps.

In a non-temporal ODB with in-place updating of objects, the OIDX needs only to be updated when objects are created, not when they are updated. In a log-only ODB, however, the OIDX needs to be updated on every object update. This might seem bad, and can indeed make it difficult to realize an efficient non-temporal ODB based on this technique. However, in the case of a *temporal* ODB, the OIDX needs to be updated on every object update also if using in-place updating, because either 1) the previous or 2) the new version must be written to a new place. Thus, when supporting temporal data management, the indexing cost is the same in these two approaches. We have in previous work developed several techniques that can be used to reduce the OIDX access cost.

3.2 Objects

In our approach, all objects smaller than a certain threshold are written as one contiguous object. Objects larger than this threshold are segmented into *subobjects*, and a *subobject index* (which also takes care of versioning of a large object) is maintained for each of these large objects (this is done transparently for the user/application). The value of the threshold can be set independently for different object classes. This is very useful, because different object classes can have different object retrieval characteristics.

Every object version in Vagabond has an associated object descriptor (OD), which contains the OID, physical location, commit timestamp (when the OD is not in the OIDX, the end timestamp of an object version is also included in the OD to reduce the cost of certain operations), and other administrative information. The entries in the leaf nodes of the subobject indexes are subobject descriptors (SODs). The SODs are also stored together with the subobject in the segments. The contents of an SOD include OID, physical location and write timestamp. While the timestamp in the OD denotes the commit timestamp of the actual object version, the write timestamp in the SOD is the time when the subobject was first written to a segment. This time is in general before the commit time of the actual transaction. When a subobject is moved because of cleaning, the write timestamp remains the same. The purpose of this, is to reduce the complexity of the cleaning.

It is also possible to write delta objects instead of the whole objects, in order to reduce the amount of data that has to be written to the log (note that unlike traditional systems, that only use delta objects to reduce the log writing, a delta object in a log-only database system can be an object version on its own, i.e., the complete version will not necessarily be written). More details about delta objects can be found in [6].

3.3 OID Indexing

In a traditional ODB, the OIDX is usually realized as a hash file or a B$^+$-tree, with ODs as entries, and using the OID as the key. In a TODB, we have more than one version of some of the objects, and we need to be able to access current as well as old versions efficiently. Our approach to indexing is to have *one* index structure, containing all ODs, current as well as previous versions. For this purpose, either a traditional multiversion access methods like the TSB-tree can be used, or the Vagabond temporal OID index which is optimized for OID indexing in TODBs [5].

In a temporal ODB, the OIDX has to be updated every time an object is updated. This is a potential bottleneck. In order to reduce the index update and lookup costs in temporal ODBs, the *persistent cache* (PCache) approach can be used. The PCache contains a subset of the entries in the OIDX. The goal is to have *the most frequently used ODs in the PCache*. In contrast to the main-memory cache (the OD cache), the PCache is persistent, so that we do not have to write its entries back to the OIDX tree during each checkpoint interval. This is actually the main purpose of the PCache: to provide an

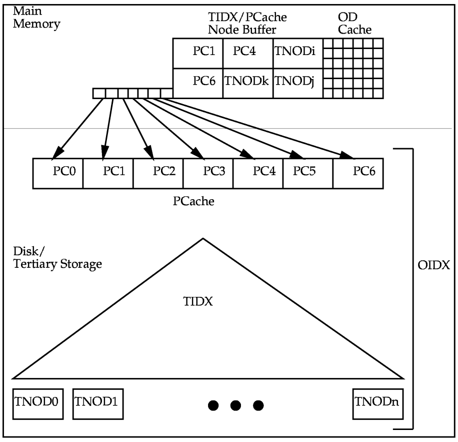


Fig. 1. Overview of the TIDX, PCache, and index-related main-memory buffers

intermediate storage area for persistent data, in this case, the ODs. The result should be a reduced update and lookup cost for ODs.

The size of the PCache is in general larger than the size of the main memory, but smaller than the size of the OIDX tree. The contents of the PCache are maintained according to an LRU like mechanism. The result should give higher locality on accesses to the PCache nodes, reducing the total number of installation reads. The average OIDX lookup cost will therefore also be less than without using a PCache.

To avoid confusion, we will hereafter denote the OID index tree itself as the *TIDX*, and use *OIDX* to denote the combined index system, i.e., the PCache and the TIDX. Thus, when we say an entry is in the OIDX, it can be in the PCache, in the TIDX, or in both. This is illustrated in Fig. 1. A more detailed description and performance analysis of the PCache can be found in [2].

4 Log-Only Database Operations

In this section we present the algorithms for the most important operations in Vagabond. We will start with an overview and an introductory example of log writing, and continue with more detailed descriptions of the different operations in the rest of the section. The description in this section is based on using magnetic disk as secondary storage. However, except for the device which stores the checkpoint blocks (see Sect. 3.1), other storage technologies can also be used.

Due to space constraints, the algorithms for vacuuming and cleaning are not described in this paper. For a detailed description of these, we refer to [6].

4.1 Introduction

When a transaction is started, it is assigned a transaction identifier (TID). Unlike many other systems, it is not necessary to write any transaction start information to the log. In fact, if a transaction is aborted before it writes any objects to the log, there will be no trace left of the aborted transaction's existence at all.

An object that is written to the log, is always written together with its OD. Writing to the log is always done by writing large segments, which in general include objects from many transactions. The buffer system employs a steal strategy, so that modified objects can be written to the log before a transaction commits. This reduces commit time, as well as making it possible to handle large amounts of data in one transaction.

When objects are written to the log before the transaction has committed, we do not know the commit timestamp. Therefore, ODs written to the log contains the TID instead of the timestamp. To be able to know if an OD contains a TID or a timestamp, TIDs always have the most significant bit set. The ODs written together with the objects in the log are only intended to be used if crash recovery is needed. In order to be able to know the timestamp of a committed transaction when doing recovery, a `(TID,timestamp)` tuple for the committing transaction is written to the log as a part of the commit.

After a transaction commits, the objects that have been created or updated by the committing transaction become current versions. When another transaction later wants to read any of these objects, it have to first retrieve the OD of the object. This OD is either still in the OD cache, or has been installed into the OIDX. During normal operation, ODs are not discarded from main memory before they have been installed into the OIDX. ODs from a committed transaction are lazily installed into the OIDX after the commit has been finished. When the ODs are inserted into the OIDX, the timestamp is used in the OD, and not the TID.

Most transactions have a short duration, and will be short enough to make it possible to write all its created ODs, and do the completion of the commit operation, during one checkpoint interval (between two consecutive checkpoints). In this way, all its ODs are guaranteed to be installed into the OIDX during the next checkpoint interval. This means that during recovery, we know that we only have to process log back to the penultimate checkpoint.

If a transaction lasts longer than one checkpoint interval, we allow ODs generated by this transaction to be inserted into the PCache, even if the transaction has not yet committed. These ODs are stored as *uncommitted ODs* in the PCache nodes, and can not be used by any other transaction. In this way, when the transaction commits, all its ODs are guaranteed to be installed into the OIDX during next checkpoint interval. Some of its ODs in the PCache will at this point still be marked uncommitted, but during crash recovery we know which committed transactions have dirty entries in the PCache, so that these can be handled properly. Entries from aborted transactions will be lazily removed from the PCache nodes as they are retrieved from disk. Objects in the log from aborted transactions will simply be discarded the next time the segments are cleaned.

Crash recovery is simple in a log-only database system. If crash recovery is needed, the last part of the log is scanned. All ODs generated from a transaction that commits during one checkpoint interval, should be installed into the OIDX when the next checkpoint interval ends, so that an upper bound exists for the amount of log that has to be processed in the case of crash recovery. ODs from committed transactions that are not yet installed into the OIDX are collected, and can later be installed into the OIDX. ODs from aborted transactions and transactions that were ongoing at crash time, are ignored.

Example. We will now illustrate log writing with the use of Fig. 2, which shows a number of transactions. On the top, we have the time-line, running from left to right, with checkpoints marked. The inserts of ODs into the OIDX is illustrated with arrows in the figure. Please note that 1) the length of a checkpoint interval depends on the workload, 2) the commit operation in the physical log is composed of suboperations (i.e., prepare, commit, and commit completed), and 3) even though the transactions are illustrated with separate lines, objects and ODs from different transactions can be stored in the same segments.

Starting with transaction T_8, this is an ordinary short transaction. The transaction commits during the second checkpoint interval, and the ODs generated from this transaction are guaranteed to be installed into the OIDX when checkpoint 4 ends. In the case of a crash after checkpoint 4, no ODs from transaction T_8 will exist in the part of the log that have to be processed during recovery.

Transaction T_7 spans more than one checkpoint interval, and is therefore treated as a long transaction. All the ODs written by transaction T_7 during the first checkpoint interval, before checkpoint 2, must be installed into the PCache before the end of checkpoint 3. This will be done in the background during the second checkpoint interval. After the transaction have committed, its ODs can be inserted into the TIDX as well. To emphasize: Before transaction T_7 commits, its ODs can only be inserted into the PCache, but after the transaction has committed, its ODs can be inserted into the TIDX as well as the PCache.

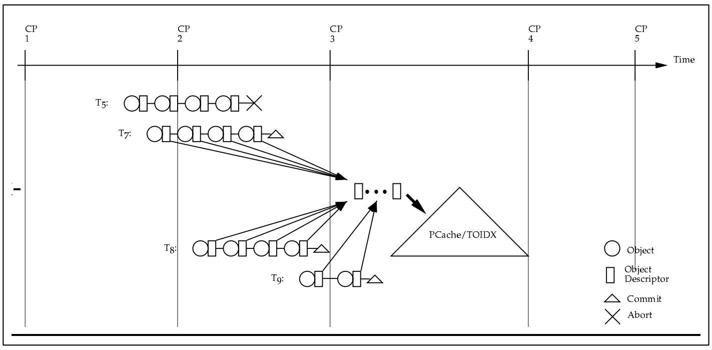


Fig. 2. Example of log writing

Similar to the case of transaction T_7, some of the ODs from transaction T_5 written during the first checkpoint interval might have been inserted into the PCache before it aborts. If this was the case, they will be removed from the PCache in a lazy way, as time goes by. ODs (and objects) written to the log will be removed later, during the segment cleaning process.

Transaction T_9 commits during the third checkpoint interval, and its ODs might be inserted into the OIDX during the same interval, but this is not guaranteed to be finished until checkpoint 5 finishes, i.e., the second checkpoint after its commit.

We have now given a short introduction to the log generation, and we will now continue with a more detailed description of the operations.

4.2 Object Operations

A new OD is created every time an object is created, updated or deleted. In the case of an object create or update, the OD is written together with the object to the log, in the case of an object delete, it will be written to the log at a convenient time. In all cases, they will be written to the log before the transaction can finish the commit operation. The ODs will be inserted into the OIDX if the transaction commits. An OD will *never* be inserted into the TIDX itself before the actual transaction commits, but in the case of large transactions, some of the ODs can be inserted into the PCache (this will be described in more detail later in this section). Modified OIDX nodes will in general be written at a later time, so that the response time for a transaction commit can be short.

The following description of the operations is mainly independent of which concurrency control strategy is used. This means that we assume that in addition to performing the actions described below, concurrency control aspects are maintained. For example, if two-phase locking is used, we expect that the necessary lock(s) have been acquired before the actual operation is carried out.

OD Management. Most transactions are short and update only a few objects. The ODs they have generated are usually not discarded from main memory until they have been installed into the OIDX. The ODs written together with the objects in the log will only be used if crash recovery is needed. All ODs resulting from a transaction committed during one checkpoint interval, should be installed into the OIDX before the next checkpoint interval ends.

In the case of long[1] transactions, the number of ODs can be very high. If we require all ODs to be resident in the OD cache until they are inserted into the OIDX, the maximal size of a transaction would in this case be restricted by the size of the OD Cache. Another problem is transactions that last longer than a certain number of checkpoint intervals. *All* of the log created from the time when this transaction started to write to the log has to be processed during crash recovery. This can take a lot of time and it makes cleaning more complex, which in many situations is not acceptable.

The problem with long transactions is solved by allowing ODs from uncommitted transactions to be inserted into the PCache (the reason for doing this, and other alternatives, are discussed in [6]). The extra cost is only marginal because PCache nodes are frequently read and written, and because most transactions commit, the PCache space wasted by this approach is low.

We do not allow ODs from uncommitted transactions to migrate further from the PCache to the TIDX. The reason for this, is that this would complicate commit processing, recovery and it would also be costly. Also, it should not be necessary. In the case of very long transactions, the size of the PCache can be adaptively resized, so that its size does not limit the transaction size.

[1] When we here study long transactions, we mean both traditional long-living transactions, as well as “large transactions” (transactions that generate large amounts of data).

Management of ODs from Uncommitted Transactions. In order to avoid other transactions accessing ODs from the uncommitted transactions, and because ODs from uncommitted transactions contain TIDs instead of timestamps, we need to know which ODs in the PCache nodes are ODs from committed transactions, and which ODs are from uncommitted transactions. This is achieved by using two binary trees inside the PCache nodes. One tree, the *committed tree*, is used for the ODs of committed transactions, and another tree, the *uncommitted tree*, is used for ODs from uncommitted transactions.

When a transaction commits, the ODs it generated, and which have been inserted into PCache nodes, should be moved from the uncommitted trees to the committed trees. This is done lazily. Every time a PCache node is retrieved, all ODs from committed transactions that are still in the uncommitted tree are moved to the committed tree. When an ODs is moved, the TID in the OD is replaced with the commit timestamp of the transaction that generated the OD.

We have to keep the TID of a committed transaction until all ODs stored in the PCache before it committed, have been moved to committed trees. For each committed transaction, we maintain a counter of how many ODs it still has in uncommitted trees in the PCache, and every time we move an OD from an uncommitted tree to a committed tree, we decrement this counter. When the counter reaches zero, we can discard information on this transaction.

The `(TID,timestamp,counter)` tuples are stored in a TID/timestamp/counter table (TTCT). The TTCT is written to the log during each checkpoint interval, in order to make it possible to reconstruct the table during recovery. Most transactions will be small, and ODs from these transactions will not be written to the PCache. Therefore, the size of this table will be relatively small.

Creating and Updating Objects. When we create an object, it is allocated a unique OID, and an OD is created. A new OD is also created every time we update an object. The OD of the new version will eventually make its way into the OIDX if the transaction commits, as described previously. There is little difference between temporal and non-temporal objects in the case of create and update operations, the difference is mostly whether the old OD is kept in the OIDX.

To ensure durability, an object *has* to be written to disk before its transaction commits. This is similar to a traditional system, where we need to have the necessary information written to the log before the transaction commits. The difference, however, is that in Vagabond, the log is the final place for the objects.

The buffer system employs a steal strategy, which means that a modified object can be written to disk before the transaction commits. The object and its OD is written together with other objects and their ODs, possibly from other transactions, into a segment. We do not know the timestamp of a transaction before it commits, so in the log we always use the TID instead of the timestamp in the OD. This is also the case for ODs of uncommitted transactions when they are inserted into the PCache.

When large objects are updated, only the modified subobjects and the affected subobject-index nodes are written to the log. The use of the Vagabond large-object index ensures that only the affected parts of the subobject index need to be written. Large objects are possibly spread over several segments, and therefore the writing of large objects has to be done carefully. This is done by first writing the updated subobjects, and then the modified parts of the subobject index. Note that the timestamps in SODs (see Sect. 3.2) are the time of segment write, not the commit time. The reason for this, is that in this way we do not have to update the subobject after the transaction has committed. The timestamps in the SODs are only used during cleaning, and for that purpose, the exact commit timestamp is not needed.

Deleting Objects. Temporal objects are not physically deleted. In this respect, they are mostly treated as non-deleted objects. For example, during cleaning, a deleted object will be moved to the new segment, similar to any non-deleted object. A non-temporal object, on the other hand, can not be accessed after it has been deleted. It will be physically removed from the segment it resides in the next time the segment is cleaned. Whether an object is temporal or non-temporal, is stored in the object's OD.

Deleting Temporal Objects. Deleting an object which is defined as temporal, is done by writing a tombstone OD, which is an OD where the physical location is NULL, and the timestamp is the delete time.

Note that whether the object is a large object or not, does not matter for the delete operation when the object is temporal.

Deleting Non-Temporal Objects. If we do not want to keep the deleted version, i.e., it is not a temporal object, its OD is written to the log with both physical location and timestamp set to NULL. Unlike the tombstone OD, this OD is written to the log as logging information to be used in the case of recovery. When the OIDX is updated later, the OD for this object will be removed. When the object is deleted, the live-byte counter (in the SST, see 3.1) for the segment where the object resides, is decremented accordingly.

If the object is a large object, the subobject index has to be traversed in order to decrease the live-byte count for the segments where the subobject-index nodes and the subobjects are stored. The subobject-index nodes and the subobjects will be deleted next time the respective segments are cleaned.

Deleting New Objects. If an object is deleted by the same transaction that created it, the effect on the database should be the same as if the object had never been created. This is assured by using the following algorithm:

1. If the object has not yet been written to the log, the only action needed is to remove the OD from the OD cache and delete the object from the main-memory buffer.
2. If the object has been written to the log, but its OD is still in the OD cache and is dirty with respect to the OIDX, the only action needed is to write a tombstone OD to the log. If a crash occurs, the recovery algorithm will know that an object deleted by the same transaction that created it, should be discarded.
3. If the object has been written to the log, and the OD in the OD cache is clean or the OD is not resident in the OD cache, that means that the OD has been inserted into a PCache node. In this case, the OD has to be removed from the PCache node. In order to avoid a synchronous operation, a tombstone OD is created and inserted into the

OD cache. The OD in the PCache node is removed the next time the PCache node is brought into main memory. The tombstone OD that is inserted into the OD cache has to be written to the log before or during commit, if the PCache node has not been updated before that time.

Reading Objects Stored as Complete Versions. We now describe how to retrieve current as well as historical object versions. The physical location of an object version is stored in its OD, and in order to read an object that is not resident in memory, we have to first retrieve its OD. If the object is a large object, the location in the OD is the location of the root of the subobject index of the actual object version, and this subobject has to be traversed in order to retrieve the requested subobject(s).

Current Versions. When we want to read the current version of an object, we first check if the OD is resident in the OD buffer. If not, we have to do a lookup in the OIDX. An OIDX lookup is done by first searching the PCache, and if the OD is not found in the PCache, the TIDX is searched. When doing a lookup in a PCache node, it is only necessary to search the uncommitted tree in the PCache node if it is possible that the object has been previously modified by the same transaction that is now requesting the object. If a locking protocol is used, this can only be the case if the transaction already owns a write lock on this object (or in the case of hierarchical locking, a lock for a larger granularity, for example a container).

Historical Versions. If we know the timestamp of the historical version we want to retrieve, the lookup for the OD and retrieval of the object can be done in the same way as when reading the current version of an object. However, quite often the query is for an object version valid at a certain time t_j. In this case, we have to retrieve the OD with *the largest timestamp less than or equal to* t_j. It is this operation that makes an end timestamp in the OD beneficial when the OD is outside the TIDX (see Sect. 3.2). If we did not have the end timestamp in the OD, it would not be sufficient to access the OD cache or the PCache to find the OD. Even if we found an OD in the OD cache or PCache with a timestamp t_i that is close to t_j, there could have been updates between t_i and t_j. This would be impossible to know from the ODs alone, and we would have to do a lookup in the TIDX for every such retrieval.

4.3 Transaction Management

In order to be able to do recovery after a failure, it is necessary to ensure that enough information has been written to the log before a transaction commits. We have previously described how objects can be written to disk before a transaction commits, in order to avoid writing all the objects modified by the transaction in one burst during commit, and how we write the ODs to the log in order to avoid synchronous updates of OIDX nodes at commit time. We will now describe in more detail transaction management in Vagabond.

Commit. The transaction commit operation can in principle be implemented by writing objects from the transaction that is still dirty in the object buffer to the log, followed by a transaction finished mark which includes a `(TID,timestamp)` tuple. After the objects and the transaction finished mark has been written to the log, the transaction commit is considered finished. Objects, ODs, and the transaction finished mark can be stored in the same segment, and more than one transaction can be committed in one segment write (similar to traditional group commit). In this way, the response time can be as low as the time it takes to write one segment, and this technique should in most cases give good throughput in a single server system. However, it is difficult to implement an efficient 2-phase commit operation by using this simple technique. 2-phase commit is crucial in a multi-server system, and in the Vagabond approach we use a more elaborate commit protocol, where more information is written to the log in the various phases of the commit process [6].

Abort. In a log-only database, we do not need to undo operations when we abort a transaction. If the transaction that wrote an object does not commit, an object written to the log before the abort operations will simply be a dead object, which will be removed the next time the segment is cleaned.

No ODs reflecting updates from a transaction will be installed into the TIDX until after the commit has been done. ODs with OIDs that have been allocated by a transaction that has aborted will never be inserted into the TIDX, and the OIDs are not reused later by any other transaction. ODs from aborted transactions that have been inserted into the PCache will be removed lazily at the same time as ODs from committed transactions are moved to the committed tree (see Sect. 1).

When a transaction aborts, the live-byte counts for the segments where the objects were written are decremented accordingly. This can only be done immediately for the objects where ODs are still in main memory. For those objects where the ODs have been removed from the OD cache and inserted into the PCache, the live-byte counts are decremented when the ODs are removed from the PCache nodes.

4.4 Recovery

When the system is restarted, it is determined from the checkpoint block whether the shutdown of the system was done controlled, or caused by a crash. If caused by a crash, recovery is needed. We now describe how to reduce the recovery time by checkpointing, and take a closer look at recovery and how to handle media failures.

Checkpointing. The main purpose of checkpointing is to reduce the recovery time. This is achieved by bounding the amount of log that has to be processed at recovery time. In a traditional database system, the main part of the checkpoint process is to write dirty pages back to disk, usually by the use of a fuzzy checkpointing technique. In a log-only system, this is not the important issue. The log is the final repository, and objects will have to be written to the log before commit in any case. In Vagabond, the main issue of checkpointing is to install the ODs into the OIDX. This is done to avoid having to read excessive amounts of log at recovery time (in order to find ODs that have not been installed into the OIDX before the system crashed).

During a checkpoint interval, between the checkpoints, ODs from committed transactions are installed into the OIDX. To keep the installation rate high enough and reduce the amount of memory needed to store ODs not yet installed into the OIDX, all ODs from a transaction committed during one checkpoint interval, should be installed into the OIDX no later than the end of the next checkpoint interval.

The segment status table (SST), PCache Status Table (PCST), and the TID/timestamp/counter tables (TTCT), are resident in main memory, and in order to be able to recreate these after a crash, the contents of these tables are written regularly to the log. This is done by writing a certain range of entries from these tables each time we write a segment. During each checkpoint interval, all SST, PCST and TTCT entries should have been written at least once.

Checkpointing can be costly, so it is important that the amount of data to be written at checkpoint time is as low as possible, and that data structures locked as a part of the checkpointing process are locked only for a short time. In Vagabond, most operations can run as normal during checkpointing. The only restriction is that the timeout values for 2-phase commit should be smaller than one checkpoint interval. The checkpoint algorithm is as follows:

1. Wait until the number of written objects since last checkpoint, or the number of segments written since last checkpoint, reaches a certain threshold.
2. If there are ODs created before the last checkpoint that are not yet installed into the OIDX, stop all other log processing until this has been done. Note that this delay is undesirable, and can normally be avoided by giving high enough priority to OIDX updating. To reduce a possibly long checkpointing time when this situation occurs, it is possible to solve the problem temporarily (or rather postpone the problem) by simply writing to the log the dirty ODs that have not been written since last checkpoint.
3. If there are entries in the SST, PCST, or TTCT, that have not been written during this checkpoint interval, write them to the log now.
4. Update the least recently written checkpoint block. Now the checkpointing is finished, and normal operation continues.

Crash Recovery. The purpose of crash recovery is to reconstruct a consistent state. In a traditional system, this is a very complex operation, and typically involves an analysis phase, a redo phase, and an undo phase. In a no-overwrite system like Vagabond, undo or redo of objects is not necessary. However, ODs and transaction management information is written to the log, and this information has to be read in order to rebuild the resident structures.

The first step in a recovery is to identify the last segment that was successfully written before the crash. This is done by reading the log from the last checkpoint until 1) we come to a segment that was only partially written (the system crashed when it was writing this segment), or 2) the *next segment* of a segment does not exist (the address of the next segment is determined before a segment is written so that it can be stored in the currently written segment, in this way, if the next segment does not exist we know that the system crashed in the interval between writing two segments).

When we read the log in order to find the end of it, we also collect all ODs, and keep each OD where we later find a commit operation for the transaction that generated that OD. For ODs that we do not find a commit, these ODs can be safely discarded because the system crashed before the transaction was committed.

After we have identified the end of the log, and processed the part of the log written after the last checkpoint, we read the log from the last checkpoint and backwards until the penultimate checkpoint. In this way, we process all segments that might have ODs from committed operation that have not yet been installed into the OIDX. This backward reading can be done efficiently, because all segments have a pointer to the previous segment. In addition, all segments also contains the number of ODs in the previous segment in the log. This makes it possible to only read the part of the segment that contains the ODs, the rest of the segment can be skipped.

While reading the segments, we rebuild the relevant structures in memory. If we need to write index nodes during recovery because of insufficient buffer capacity, this is done to clean segments. When the log has been processed, we do a checkpoint, and when the checkpoint process is finished, the checkpoint blocks are updated. Idempotence is guaranteed because we do not modify any written data before updating the checkpoint block. If a system crashes during recovery, it will simply start recovery in the same way next time.

Media failure in a log-only system can be handled by the use of mirroring (RAID 1) or RAID with parity blocks (for example RAID 4 or RAID 5). The use of mirroring will also improve read performance, because the read bandwidth is doubled. The write performance will stay the same.

5 Conclusions and Further Work

The log-only approach for TODBs has been shown to be promising with respect to performance [3]. However, although the log-only approach in its basic form is relatively straightforward, achieving performance and high concurrency is not straightforward. In this paper, we have described in detail how this can be achieved, resulting in a design that avoids some of the problems in previous, related designs. The results include support for steal/no-force buffer management, fuzzy checkpointing and fast commit.

The results from this paper together with our other work in this context, has shown that the log-only approach is feasible. Some ideas that we would like to explore further are the realization of a valid time or bitemporal TODBs using the log-only approach. We also plan to adapt the ideas to temporal object-relational database system, which could make the results even more interesting.

References

1. J. Gray, P. McJones, M. Blasgen, B. Lindsay, R. Lorie, T. Price, F. Putzolu, and I. Traiger. The recovery manager of the System R database manager. *ACM Computing Surveys*, 13(2), 1981.
2. K. Nørvåg. The Persistent Cache: Improving OID indexing in temporal object-oriented database systems. In *Proceedings of the 25th VLDB Conference*, 1999.
3. K. Nørvåg. A comparative study of log-only and in-place update based temporal object database systems. In *Proceedings of the Ninth International Conference on Information and Knowledge Management (CIKM'2000)*, 2000.
4. K. Nørvåg. Design issues in transaction-time temporal object database systems. In *Proceedings of ADBIS-DASFAA'2000*, 2000.
5. K. Nørvåg. The Vagabond temporal OID index: An index structure for OID indexing in temporal object database systems. In *Proceedings of the 2000 International Database Engineering and Applications Symposium (IDEAS)*, 2000.
6. K. Nørvåg. *Vagabond: The Design and Analysis of a Temporal Object Database Management System*. PhD thesis, Norwegian University of Science and Technology, 2000.
7. M. Rosenblum and J. K. Ousterhout. The design and implementation of a log-structured file system. In *Proceedings of the Thirteenth ACM Symposium on Operating System Principles*, 1991.
8. V. Singhal, S. Kakkad, and P. Wilson. Texas: An efficient, portable persistent store. In *Proceedings of the Fifth International Workshop on Persistent Object Systems*, 1992.
9. M. Stonebraker. The design of the POSTGRES storage system. In *Proceedings of the 13th VLDB Conference*, 1987.

Operations for Conceptual Schema Manipulation: Definitions and Semantics

Helle L. Christensen, Mads L. Haslund, Henrik N. Nielsen, and Nectaria Tryfona

Department of Computer Science, Aalborg University
Fredrik Bajersvej 7E, 9220 Aalborg Øst, DENMARK
{arnaq,levi,moth,tryfona}@cs.auc.dk

Abstract. This paper concerns the issue of conceptual schema integration and manipulation, i.e., the process of manipulating highly abstract, semantically rich database schemas for the extraction of meaningful and unambiguous results. We propose the use of six fundamental manipulation operations, namely *rename, select, project, union, set difference*, and *intersection*. We give their definition and semantics using the Entity-Relationship modeling notation. We argue that in order to preserve the semantics of the ER-schemas before and after the operations are performed, these schemas need to be translated to mathematical formulations; then the manipulation operations can be applied. We use the $\mathcal{ALCQI}$ description logic notation which features a rich combination of constructors, powerful enough to express elements of ER-schemas. We show that the resulting knowledge bases, which encapsulate all the knowledge about the ER-schemas, need further structuring to accommodate the semantics of the ER elements. Examples demonstrate the applicability and efficiency of the proposed approach.

1 Introduction

The need for *integration* and further *manipulation* in the area of databases and data warehouses has been well-documented in literature [9], [11], [12]. The process of integration relates mostly to *source integration* (i.e., when the sources are merged), *schema integration* (i.e., when conceptual and/or logical schemas are integrated), *data integration* (i.e., the process of comparing source data sets and creating a reconciled view), and *information integration* (i.e., when metadata or views are merged), while the process of manipulation includes issues such as pre-integration, schema comparison, schema union, schema intersection, and schema restructuring [10].

In this paper we deal with conceptual schema manipulation, i.e., the process of manipulating highly abstract, semantically rich schemas with set operations and operations comparable to the ones from relational algebra, for the extraction of meaningful results. These schemas may represent either different applications of the same domain, e.g., a patient and a drug trial schema for the corresponding databases of a pharmaceutical company, or just different excerpts of the same database.

The motivation behind this research effort is based on the argument that in order to build applications out of existing ones we need to raise the level of abstraction of operations on schemas, models and mappings [1]. Our goal is a methodology that supports conceptual schema manipulation in an unambiguous manner.

More specifically, in this work, having the Entity-Relationship (ER) as the pilot conceptual model, we introduce the semantics and usage of six *schema manipulation operations*, namely *rename, select, project, union, set difference*, and *intersection*, for the manipulation of ER-schemas. The semantic definition of these operations is not enough to ensure their unmistakable application on conceptual schemas. A mathematically sound formulation, which captures the semantics of both the ER-schemas to be manipulated and the applied operations in a formal way, is needed. For this, we use Description Logic (DL) for the unambiguous representation of ER-schemas, as it is particularly well-suited for specifying data classes and relationships among classes and equipped with both formal semantics and inference engines [5]. The DL we use, called $\mathcal{ALCQI}$, for the representation of the ER-schemas, features a rich combination of constructors powerful enough to express elements of ER-schemas. The expressions of $\mathcal{ALCQI}$ form *Knowledge Bases* (KB) about classes and relations, which ensure the preservation of the semantics for both ER-schemas and the applied operations.

The rest of the paper is organized as follows. Section 2 describes representative related work on the issues of schema integration and manipulation as well as the use of DL in this area. Section 3 gives the semantics of the schema manipulation operations and shows their use on ER-schemas. Section 4 describes the way DL is used to preserve the semantics of the ER-schemas to be manipulated. The presented approach has been tested by manipulating the Data Management (*DM*) and the International Product Safety (*IPS*) database schemas of the Danish pharmaceutical company Novo Nordisk A/S; examples from this area demonstrate the applicability of our proposal. Section 5 concludes and draws future research directions.

2 Related Work

The approaches that have been developed in the area of integration differ with respect to the merging of elements of the used models. Moreover, the use of DL, to ensure preservation of semantics during manipulation, has been broadly adopted in the area of integration.

According to [2] three types of description logic assertions can be used to determine relations between elements. Elements can only be merged if they fulfill the requirements set by the user, e.g., elements must be 85% equal to be merged. This process of creating a global view over a database might create conflicts e.g., homonym conflicts where entities have the same name but are interpreted differently. In [4] a solution on how data integration conflicts can be resolved through suitable matching, reconciliation, and conversion operations is explained. The solution is based on a conceptual representation of the application domain which helps in the integration and reconciliation process.

The research areas of integration and manipulation are explored in different directions. According to [3] two main approaches for integration exist: the procedural and the declarative. In the procedural approach data is integrated in an ad-hoc manner with respect to some predefined requirements. Several projects follow this approach, such as the TSIMMIS, *Squirrel*, and WHIPS, which all are explained briefly in [10]. The declarative approach on the other hand works towards modeling data in a suitable language to construct a unified representation. There are also tools that supports this approach, such as the *Carnot*, SIMS, and Information Manifold which are also explained briefly in [10]. Another way to facilitate the process of integration is presented in [8] which describes a graphical tool called i•com. This tool facilitates the process of conceptual modeling, integration of databases, and uses a description logic inference engine to check possible consistencies of the created schemas.

Our work presents a methodology to manipulate ER-schemas in an unambiguous way with the help of DL. We do not aim towards the implementation of a graphical tool, like i•com; we rather envision a mechanism which can be included on top of an already implemented system.

3 The Schema Manipulation Operations

In order to efficiently manipulate ER-schemas, we introduce the use and semantics of six fundamental operations. These operations, namely *rename, select, project, union, set difference*, and *intersection*, were chosen having as basis the fundamental operations from the relational algebra. They are constructed based on a set of guidelines that specifies choices made about integration of ER-schemas, for example, rules for merging schema elements, i.e., entities, relationships, attributes. (We chose to use the terms entity, and relationship, instead of entities, and relationships, respectively).

The merging of schema elements during the manipulation process is based on the concepts of *entity-equivalence*, *attribute-equality*, and *relationship-equality*. More specifically, we allow the merging of two entities from two ER-schemas if they have the same name or if the user specifies them to be the same (entity-equivalence); we allow the merging of two attributes if they belong to the same entity and have the same names (attribute-equality). Two relationships are equal (relationship-equality) if they relate the same entities with the same cardinalities.

Additionally, we make the assumptions that the ER-schemas to be integrated and manipulated are logically connected, i.e., belong to the same domain and concern the same topics, e.g., sales, production, or patients. There exist cases which are difficult to address, such as synonym cases, e.g., entities may have synonym names and only some of their attributes are the same. We handle them by allowing the user to specify equivalent entities and ISA-relationships between entities in different ER-schemas. The notation used to specify equivalence between entities is $Entity_1(S_1) =_{SPE} Entity_2(S_2)$, which denotes that $Entity_1$ from the

ER-schema S_1 is specified to be the same as $Entity_2$ from the ER-schema S_2. The notation used to specify a ISA-relationships between entities in different ER-schemas is $Entity_1(S_1) \sqsubseteq_S Entity_2(S_2)$, which denotes that $Entity_1$ from the ER-schema S_1 is a specialization of $Entity_2$ from the ER-schema S_2.

3.1 An Application Example

We tested the applicability of our approach in excerpts of two ER-schemas used in the pharmaceutical company Novo Nordisk A/S. The two schemas correspond to the Data Management (DM) database which keeps track (among others) of products, trials, patients, and events, and the International Product Safety (IPS) database which records serious events. The products are tested in trials involving test persons, which are referred to either as *patients* or as *subjects*. During a trial a patient can suffer adverse *events*. An event can be e.g., a slight bruise, an allergic reaction, or death. A subset of these events is *serious adverse events*. A serious adverse event can be e.g., anaphylactic shock or death. The IPS ER-schema regards data about the adverse serious events while the DM ER-schema regards data about all events. Figures 1 and 2 illustrate excerpts of the two schemas.

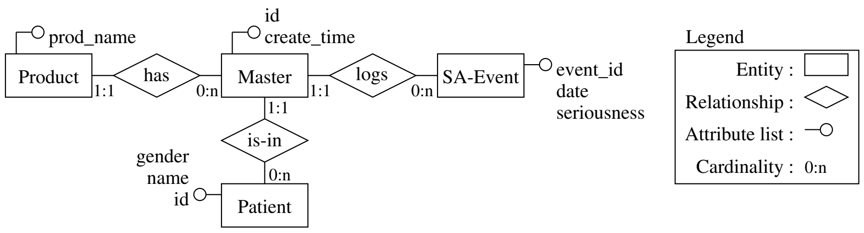


Fig. 1. *The IPS ER-schema*

It is obvious that the two database schemas have many common values in basic elements (e.g., patients, drugs, etc.) and the need for their integration is apparent. As described previously, the user can specify equivalent entities and ISA-relationships in the two ER-schemas. The following shows the specifications of equivalent entities and ISA-relationships for the two ER-schemas:

– $Master(IPS) =_{SPE} Trial(DM)$, $Patient(IPS) =_{SPE} Subject(DM)$
– $SA\text{-}Event(IPS) \sqsubseteq_S Event(DM)$

3.2 The Semantics of the Six Schema Manipulation Operations

The schema manipulation operations operate on a universe of discourse, which we define to be the set Ω that consists of all ER-schemas. We further define the

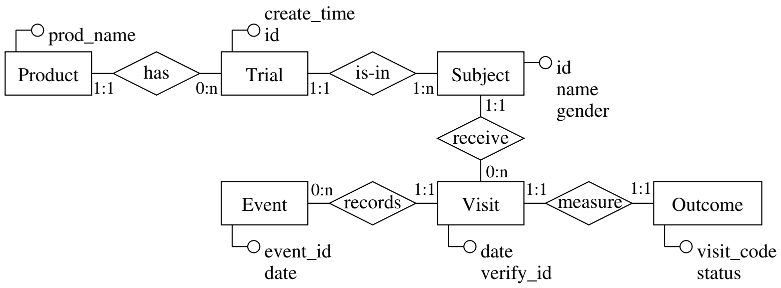


Fig. 2. *The DM ER-schema*

set $\mathcal{P}$, that consists of all predicates P. With these definitions, it is possible to define the domain and range of the schema manipulation operations. The domain and range for the six operations is summarized in Table 1a. A description of the semantics of the six operations follows.

The *Rename* Operation. The *rename* operation receives an ER-schema and a predicate as input and returns a new ER-schema; the predicate is a comma separated list containing tuples, where each tuple contains the current name (*fromName*) of the entity and its new unique name (*toName*). The new ER-schema contains all the entities which were in the original ER-schema. The entities which were specified to be renamed, are renamed according to the used predicates. The semantics of the *rename* operation are summarized in Table 1b. An example of the *rename* operation is that of renaming the entity *Subject*

Table 1. (a) Domain and range for the operations. (b) The *rename* operation

Operation	Domain & Range
rename	$\Omega \times \mathcal{P} \rightarrow \Omega$
select	$\Omega \times \mathcal{P} \rightarrow \Omega$
project	$\Omega \times \mathcal{P} \rightarrow \Omega$
union	$\Omega \times \Omega \rightarrow \Omega$
set difference	$\Omega \times \Omega \rightarrow \Omega$
intersection	$\Omega \times \Omega \rightarrow \Omega$

(a)

```
$Rename_P(S_1) \rightarrow S_2$

 1. Copy $S_1$ to $S_2$
 2. For each tuple in predicate $P$ do:
     a) Replace all occurrences of
        fromName in $S_2$ with toName.
```

(b)

in the *DM* ER-schema, which according to the specification in Sect. 3.1 can be renamed to *Patient*. The syntax for the example, where entities in the *DM* ER-schema are renamed, is: $rename_{((Subject,Patient))}(DM)$.

The *Select* Operation. The *select* operation receives an ER-schema and a predicate as input and returns a new ER-schema; the predicate consists of a list of names, specifying the entities to be selected. The resulting ER-schema consists of the entities specified in the predicate and any entities related to the entities specified in the predicate. The semantics of the operation are summarized in Table 2a. An example of its use is illustrated in Fig. 3a.

The *Project* Operation. The *project* operation receives an ER-schema as input and returns a new ER-schema that consists of the entities specified in the predicate and the relationships relating these entities. The semantics of the *project* operation are summarized in Table 2b. An example of the use of the *project* operation applied to the ER-schema of Fig. 1 is illustrated in Fig. 3b.

Table 2. (*a*) The *select* operation. (*b*) The *project* operation

$Select_P\ (S_1) \rightarrow S_2$

1. For each tuple in predicate P do:
   a) Copy the entity along with all entities related to the entity from S_1 to S_2 (ignore entities already in S_2).

(*a*)

$Project_P\ (S_1) \rightarrow S_2$

1. Copy all entities specified in the predicate P from S_1 to S_2.
2. Copy all relationships in S_1 which in S_2 are related to two or more entities.

(*b*)

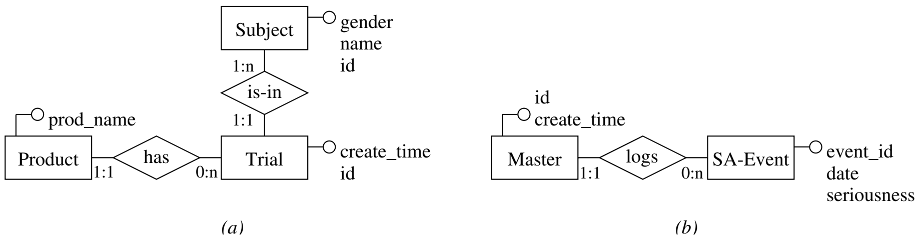


Fig. 3. *(a)* $select_{Trial}(DM)$. *(b)* $project_{Master,SA-Event}(IPS)$

The *Union* Operation. The *union* operation receives two ER-schemas as input and returns a new ER-schema containing the two original schemas merged together. Equivalent entities are merged into one entity with a union of the attributes of the original entities. It is common, when unifying schemas, to have two entities, one in each schema, which are semantically related by the ISA-relationship. Users can specify ISA-relationships between entities in the two schemas as mentioned in Sect. 3.1. This information is further stored in a list. This list, called the *ISA-list*, contains *ISA-tuples*, each specifying an ISA-relationship between entities from two ER-schemas. When two ER-schemas, that have an ISA-relationship specified between entities, are merged, a new relationship is added. This new relationship connects the two entities, from the ISA-relationship, in the new ER-schema. The semantics of the *union* operation are summarized in Table 3a and an example of use of the *union* operation can be seen in Fig. 4. The relationship *isa** represents the relationship which is added

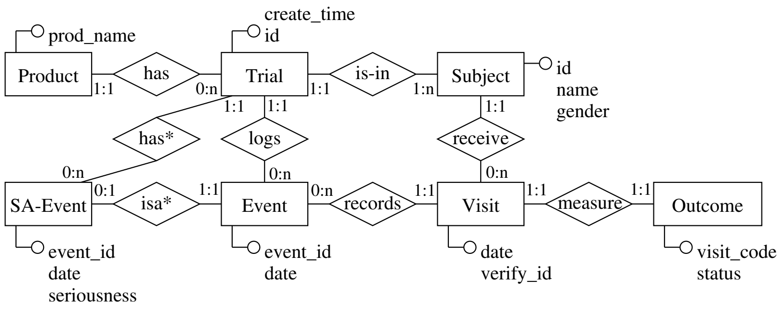


Fig. 4. *union*(*DM*, *IPS*)

in item 7a of the semantics of Table 3a, while the relationship *has** represents the *has* relationship from the *IPS* schema, which has been renamed to be unique.

The *Set Difference* Operation. The *set difference* operation receives two ER-schemas as input and returns a new ER-schema consisting of all the entities in the ER-schema S_1 minus the entities that exist in the ER-schema S_2. This implies that the operation simply removes the intersection of entities between the two ER-schemas from the first ER-schema. The semantics of the *set difference* operation are summarized in Table 3b and an example of the *set difference* operation can be seen in Fig. 5a. Note that the *set difference* operation is not commutative.

The *Intersection* Operation. The *intersection* operation receives two ER-schemas as input, and returns a new ER-schema consisting of the entities and relationships which exist in both of the original ER-schemas. The semantics of the *intersection* operation are summarized in Table 3c and an example of use of the *intersection* operation can be seen in Fig. 5b. Note that the *intersection* operation takes the union of attributes of each common entity and adds then to the entity appearing in the new ER-schema.

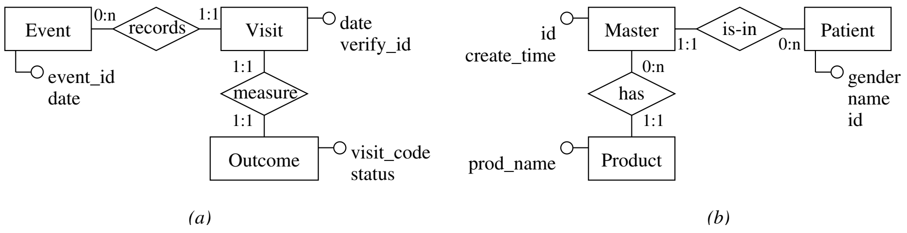


Fig. 5. *(a) set difference*(DM, IPS). *(b) intersection*(IPS, DM)

Table 3. (a) The *union* operation. (b) The *set difference* operation. (c) The *intersection* operation

Union $(S_1, S_2) \rightarrow S_3$

1. Copy S_1 to S_3
2. Copy S_2 to S_4
3. Find all relationships in S_3 and S_4 that have the same name, but are not equal.
   a) Change the names of relationships found in S_4, so they are unique in both S_3 and S_4.
4. Find all entities from S_3 and S_4 that are equivalent.
   a) Add unique attributes from entities in S_4 to entities in S_3.
5. Add unique entities from S_4 to S_3.
6. Add unique relationships from S_4 to S_3.
7. For each tuple in the ISA-list do:
   a) Add a new relationship to S_3.

(a)

Set Difference $(S_1, S_2) \rightarrow S_3$

1. Copy S_1 to S_3
2. Remove all entities from S_3 which have equivalent entities in S_2.
3. Remove all relationships which belong to zero or one entities, along with any references to these relationships, from S_3.

(b)

Intersection $(S_1, S_2) \rightarrow S_3$

1. Find all equivalent entities in S_1 and S_2
   a) Add union of the two entities to S_3.
2. Add all equal relationships from S_1 and S_2 to S_3.

(c)

4 Description Logic and the Manipulation Operations

In order to ensure that the application of the manipulation operations to schemas returns unambiguous results, we need to define the ER-schemas in a clear, formal way, with unique interpretations. For this, we adopt the use of the description logic $\mathcal{ALCQI}$ and the translation of ER-schemas to KBes, which is described in [5], although the notation is inspired by [7]. The reason for choosing $\mathcal{ALCQI}$ is that it provides a broad formal framework, which is useful for the preservation of semantics and the expression of the ER-schemas elements. We first give a brief introduction to $\mathcal{ALCQI}$ (Sect. 4.1) and then describe the mapping and translation of ER-schemas to $\mathcal{ALCQI}$ KBes (Sect. 4.2 and 4.3). For further preservation of semantics, the KBes need to be structured to accommodate the elements of the ER-schemas (Sect. 4.4).

4.1 The Description Logic $\mathcal{ALCQI}$

The basic types of $\mathcal{ALCQI}$ are *concepts* and *roles*. Concepts and roles belong to a domain Δ, which consists of a set of instances. The domain Δ could e.g., be *Citizen of Denmark*, where the instances are people living in Denmark.

A concept is a subset of the domain Δ and it can be either *atomic* or *composed*. Composed concepts are expressions built by applying *concept constructors* on a set of atomic concepts. An atomic concept is a class in the domain Δ and it can be chosen to be any of the classes in the domain, but some of the classes in the domain are more useful as atomic concepts than others. A concept in the domain *Citizen of Denmark* could be *male*, which consists of all male people in the domain, whereas a composed concept could be *female* $= \neg$*male*.

A role is a binary relation between concepts of the domain Δ and is, like a concept, either atomic or composed. Composed roles are expressions made by applying *role constructors* to a set of atomic roles. A role in the domain *Citizen of Denmark* could be *parent_of*, which relates all parents in the domain *Citizen of Denmark* with their children. A composed role could be *child_of* $= \neg$ *parent_of*, which relates all children with their parents.

The syntax for the constructors (both concept and role constructors) is shown in the syntax part of Table 4, where C and C' are concepts and R is a role. The semantics of $\mathcal{ALCQI}$ are given as sets on the domain Δ. A concept C is interpreted as a set of instances on the domain Δ, while a role is interpreted as a set of pair of instances of the domain Δ. Formally, an *interpretation* $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ consists of a nonempty set $\Delta^{\mathcal{I}}$ and a function $\cdot^{\mathcal{I}}$ that maps every concept to a subset of $\Delta^{\mathcal{I}}$, every role to a subset of $\Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$ and every individual to an instance of $\Delta^{\mathcal{I}}$. The set $\Delta^{\mathcal{I}}$ is called the domain, while $\cdot^{\mathcal{I}}$ is called the interpretation function. The semantics of all the constructions are shown in the semantics part of Table 4.

The constructors in Table 4 are used to form expressions in $\mathcal{ALCQI}$. A set of expressions in $\mathcal{ALCQI}$ forms a KB. A KB encapsulates knowledge about a specific domain, expressed in terms of concepts and roles. An $\mathcal{ALCQI}$ KB is constituted by a finite set of assertions. There exist two types of assertions in $\mathcal{ALCQI}$: *inclusion* and *equality assertions*. The assertions consist of atomic concepts A and C, where C can be an atomic or composed concept. These assertions represent each knowledge about the domain. The syntax of the inclusion assertion is $A \sqsubseteq C$, while the semantics is $(A \sqsubseteq C)^{\mathcal{I}} = (A^{\mathcal{I}} \subseteq C^{\mathcal{I}})$. The syntax of the equality assertion is $A \doteq C$, while the semantics is $(A \doteq C)^{\mathcal{I}} = (A^{\mathcal{I}} = C^{\mathcal{I}})$.

Table 4. The syntax and semantics rules of $\mathcal{ALCQI}$

Syntax		Semantics
C, C' $\longrightarrow$	A |	
	$\neg C$ |	$(\neg C)^{\mathcal{I}} = \Delta^{\mathcal{I}} \backslash C^{\mathcal{I}}$
	$C \sqcap C'$ |	$(C_1 \sqcap C_2)^{\mathcal{I}} = C_1^{\mathcal{I}} \cap C_2^{\mathcal{I}}$
	$C \sqcup C'$ |	$(C_1 \sqcup C_2)^{\mathcal{I}} = C_1^{\mathcal{I}} \cup C_2^{\mathcal{I}}$
	$\forall R.C$ |	$(\forall R.C)^{\mathcal{I}} = \{o \in \Delta^{\mathcal{I}} \vert \forall o'. <o, o'> \in R^{\mathcal{I}} \rightarrow o' \in C^{\mathcal{I}}\}$
	$\exists R.C$ |	$(\exists R.C)^{\mathcal{I}} = \{o \in \Delta^{\mathcal{I}} \vert \exists o'. <o, o'> \in R^{\mathcal{I}} \wedge o' \in C^{\mathcal{I}}\}$
	$\exists^{\geq n} R.C$ |	$(\exists^{\geq n} R.C)^{\mathcal{I}} = \{o \in \Delta^{\mathcal{I}} \vert \#\{o'. <o, o'> \in R^{\mathcal{I}} \wedge o' \in C^{\mathcal{I}}\} \geq n\}$
	$\exists^{\leq n} R.C$	$(\exists^{\leq n} R.C)^{\mathcal{I}} = \{o \in \Delta^{\mathcal{I}} \vert \#\{o'. <o, o'> \in R^{\mathcal{I}} \wedge o' \in C^{\mathcal{I}}\} \leq n\}$
R $\longrightarrow$	P |	
	P^-	$(P^-)^{\mathcal{I}} = \{<o, o'> \in \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}} \vert <o', o> \in P^{\mathcal{I}}\}$

4.2 Mapping between ER and $\mathcal{ALCQI}$

The ER model cannot be mapped directly to $\mathcal{ALCQI}$ as it consists of other types of basic elements than $\mathcal{ALCQI}$. The ER-model consists of entities, relationships, attributes, and ER-roles, while $\mathcal{ALCQI}$ consists of concepts and roles [5]. The elements from the ER-model should be modeled as DL elements to create an *$\mathcal{ALCQI}$ descriptive logic knowledge base*[5]. A KB is a formal representation that encapsulates all knowledge about a given ER-schema and enables the identification of all the original elements of the ER-schema. The basic elements of the ER model are modeled as elements of $\mathcal{ALCQI}$ (Table 5). The function $cmin_s$ maps an ER-role to a non-negative integer, whereas the function $cmax_s$ maps an ER-role to a non-negative integer in the interval from zero to infinity.

Table 5. Mapping between the ER model components and $\mathcal{ALCQI}$ components

Basic element in ER-model	Modeled as	Notes
Entity	Concept	-
Attribute	Role	Relates an entity to the domain of the attribute.
Relationship	Concept	Relationships must have unique names within the ER-schema, due to constrains in $\mathcal{ALCQI}$.
ER-role	Role	Relates a relationship and an entity.
Cardinality	Role	Uses the functions $cmin_s$ and $cmax_s$ to model upper and lower bounds on the role.

4.3 Translation from ER to $\mathcal{ALCQI}$

The translation function ϕ translates an ER-schema into a $\mathcal{ALCQI}$ KB. An $\mathcal{ALCQI}$ KB $\phi(\mathcal{S})$, representing the ER-schema $\mathcal{S}$, consists of an atomic concept $\phi(A)$ for each entity, relationship or domain symbol A in $\mathcal{S}$, and an atomic role $\phi(R)$ for each attribute symbol or ER-role R in $\mathcal{S}$.

In [5] the translation from an ER-schema to an equivalent $\mathcal{ALCQI}$ KB is done through five translation rules. Each rule translates a different part of the ER-schema into description logic expressions. Only three out of the five rules from [5] are useful in our context. These are:

1. For each entity E with attributes $A_1, \ldots, A_h$ with the domains $D_1, \ldots, D_h$ respectively, insert the assertion:
$$\phi(E) \sqsubseteq \forall\phi(A_1).\phi(D_1) \sqcap \cdots \sqcap \forall\phi(A_h).\phi(D_h) \sqcap \exists^{=1}\phi(A_1) \sqcap \cdots \sqcap \exists^{=1}\phi(A_h)$$

2. For each relationship R of arity k between entities $E_1, \ldots, E_k$, to which R is connected by means of ER-roles $U_1, \ldots, U_k$ respectively, insert the assertions:
$$\phi(R) \sqsubseteq \forall\phi(U_1).\phi(E_1) \sqcap \cdots \sqcap \forall\phi(U_k).\phi(E_k) \sqcap \exists^{=1}\phi(U_1) \sqcap \cdots \sqcap \exists^{=1}\phi(U_k)$$
$$\phi(E_i) \sqsubseteq \forall(\phi(U_i))^{-}.\phi(R), \qquad \text{for } i \in \{1, \ldots, k\}$$

3. For each ER-roles U associated to relationship R and entity E,
   - if $m = cmin_S(U) \neq 0$, then insert the assertion: $\phi(E) \sqsubseteq \exists^{\geq m}(\phi(U))^{-}$
   - if $m = cmax_S(U) \neq \infty$, then insert the assertion: $\phi(E) \sqsubseteq \exists^{\leq m}(\phi(U))^{-}$

4.4 Structuring the Knowledge Bases

Even though a KB encapsulates all knowledge about a given ER-schema and enables the identification of all the original elements, the structure is unfit for locating specific elements. The KB is therefore further structured into a *Structured Knowledge Base* (SKB). A SKB contains two lists, one that contains knowledge about entities in the KB, called the *Entity Concept list* (EC-list), and another that contains knowledge about relationships in the KB, called the *Relationship Concept list* (RC-list).

The EC-list has an entry for each entity in the ER-schema. Each entry in the EC-list consists of the name of the entity that it represents and two lists. One list, the relationship-expression list (RE-list), contains information about the relationships that the entity is related to and the other list, the domain-expression list (DE-list), contains information about the attributes of the entity.

The RC-list has an entry for each relationship in the ER-schema. Each entry in the RC-list consists of the name of the relationship that it represents and one list. The list of the relationship entry, called the relationship-expression list, contains information about the entities that the relationship is related to. The SKB for an ER-schema S has the following structure:

Structured knowledge base $\phi(S)$:
$EC = (EC_1, EC_2, ..., EC_n)$, where $EC_i = (RE_{EC_i}, DE_{EC_i})$
$RC = (RC_1, RC_2, ..., RC_m)$, where $RC_i = (RE_{RC_i})$

A KB is structured into a SKB by dividing all the assertions in the KB into *explicit assertions*. The assertions of the KB are made explicit by applying the following transformation, which can be proven by induction:

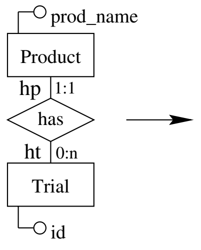
$$\mathrm{A} \sqsubseteq \mathrm{C}_1 \sqcap \cdots \sqcap \mathrm{C}_n \Leftrightarrow (\mathrm{A} \sqsubseteq \mathrm{C}_1) \sqcap \cdots \sqcap (\mathrm{A} \sqsubseteq \mathrm{C}_n)$$

Each of these explicit assertions is then grouped into the SKB based on the following three rules:

- If the concept of the constructor is represented by a domain symbol D_i, which denotes the union of all attribute domains, then the assertion belongs to the domain-expression list of the entity specified in the assertion.
- If the role of the constructor is represented by an inverse role, then the assertion belongs to the relationship-expression list of the entity specified in the assertion.
- If none of the two above rules apply, then the assertion belongs to the relationship list.

An ER-schema and its equivalent KB are shown in Table 6. The ER-schema is an excerpt of Fig. 2, with the addition of the ER-roles *ht* and *hp*. This KB can then be structured into the SKB shown in Table 7. When an ER-schema has been translated into an $\mathcal{ALCQI}$ KB, the schema manipulation operations can be applied. Each one of the operations is realized in terms of an algorithm that takes as input the appropriate KBes and returns a KB. With basis in the semantics for the schema manipulation operations and the structure of the SKB, it is a simple task to specify algorithms that implement the schema operations. The algorithms that have been created are presented in [6].

Table 6. An excerpt of the *IPS* ER-schema with the addition of ER-roles and its equivalent KB



$$\begin{aligned}
\phi(Product) &\sqsubseteq \forall\phi(product\_name).D_1 \sqcap \exists^{=1}\phi(product\_name) \sqcap \\
&\quad \forall\phi(hp))^{-}.\phi(Has) \sqcap \exists^{=1}(\phi(hp))^{-} \\
\phi(Trial) &\sqsubseteq \forall\phi(id).D_2 \sqcap \exists^{=1}\phi(id) \sqcap \forall(\phi(ht))^{-}.\phi(Has) \\
\phi(Has) &\sqsubseteq \forall\phi(hp).\phi(Product) \sqcap \exists^{=1}\phi(hp) \sqcap \\
&\quad \forall\phi(ht).\phi(Trial) \sqcap \exists^{=1}\phi(ht)
\end{aligned}$$

5 Conclusions and Future Work

We have presented the definitions and semantics of six fundamental operations for conceptual schema manipulation, namely *rename, select, project, union,*

Table 7. The SKB for the ER-schema shown in Table 6

Entity Concept	RE-list	DE-list
Product	$(\forall(\phi(hp))^{-}.\phi(has))$, $(\exists^{=1}(\phi(hp))^{-}))$	$(\forall\phi(product\_name).D_1\sqcap \exists\phi(product\_name))$
Trial	$(\forall\phi(ht))^{=1}.\phi(has)$	$(\forall\phi(id).D_2 \sqcap \exists^{=1}\phi(id)$

Relationship Concept	RE-list
Has	$(\forall\phi(hp).\phi(Product) \sqcap \exists^{=1}\phi(hp))$, $(\forall\phi(ht).\phi(Trial) \sqcap \exists^{=1}\phi(ht))$

set difference, and *intersection*, as one step towards a methodology for semi-automatic schema management. We have shown the way these operations manipulate ER-schemas. We use the ER-model as a prototype environment; we are working towards showing that object oriented [13] and other semantic models can be used as framework, once their elements and properties have been defined formally.

In order to make sure that we preserve both the semantics of the operations and the manipulated schemas, we adopt the use of $\mathcal{ALCQI}$. More specifically, we translate ER-schemas into KBes that encapsulate all their semantics and use the KBes as input to the applications of algorithms that implement the six operations. The output is a KB that represents a correct and unambiguous schema.

We have applied our approach to the real environment of the pharmaceutical databases of Novo Nordisk A/S, experiencing useful results, with no inconsistencies.

We are currently working towards the implementation of a mechanism handling the aforementioned proposal on top of the Designer and Repository tool of Oracle. We are also working on allowing semantically richer the ER-schemas, e.g., by diagrammatically allowing ISA-relationships and aggregated entities. Furthermore, we are considering the extension of the operations towards a more general use; for example, the rename, select, and project operations can also be applied on relationships if accompanied by the appropriate constraints.

Acknowledgments. The authors thank Novo Nordisk A/S for supporting this research and providing the case study environment, and Luca Aceto for his help with DL.

References

1. Philip A. Bernstein. Panel: Is Generic Metadata Management Feasible? In *Proc. of the 26th Intl. Conference On Very Large Data Bases*, pages 660–663, 2000.
2. A. Bonifati, L. Palopoli, D. Saccà, and D. Ursino. Discovering Description Logic Assertions from Database Schemes. In *Proc. of the Intl. Workshop on Description Logics - DL-97*, pages 144–148, 1997.

3. Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, Daniele Nardi, and Riccardo Rosati. Information Integration: Conceptual Modeling and Reasoning Support. In *Proc. of the 3rd IFCIS Intl. Conference on Cooperative Information Systems, New York, USA*, pages 280–291. IEEE-CS Press, August 20-22 1998.
4. Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, Daniele Nardi, and Riccardo Rosati. A Principled Approach to Data Integration and Reconciliation in Data Warehousing. In *Proc. of the Intl. Workshop on Design and Management of Data Warehouses(DMDW'99)*, volume 19, June 1999.
5. Diego Calvanese, Maurizio Lenzerini, and Daniele Nardi. *Logics for Databases and Information Systems*, chapter 8, pages 229–263. Kluwer academic publishers, 1998.
6. Helle L. Christensen, Mads L. Haslund, and Henrik N. Nielsen. Operations for Schema Integration, Definitions & Semantics. Technical report, Aalborg University, 2001.
7. Francesco M. Donini, Maurizio Lenzerini, Daniele Nardi, and Andrea Schaerf. Reasoning in Description Logics. In *Foundation of Knowledge Representation*, pages 191–236. CSLI-Publications, 1996.
8. Enrico Franconi and Gary Ng. The i•com Tool for Intelligent Conceptual Modelling. In *7th Intl. Workshop on Knowledge Representation meets Databases (KRDB'00)*, August 2000.
9. William H. Inmon. *Building the Data Warehouse*. John Wiley & Sons, 2nd edition edition, March 1996.
10. Matthias Jarke, Maurizio Lenzerini, Yannis Vassiliou, and Panos Vassiliadis. *Fundamentals of Data Warehouses*. Springer Verlag Berlin Heidelberg, 2000.
11. Peter McBrien and Alexandra Poulovassilis. A Formal Framework for ER Schema Transformation. In *Conceptual Modeling - ER '97, 16th Intl. Conference on Conceptual Modeling, Los Angeles, California, USA, November 3-5, 1997, Proc.*, volume 1331 of *Lecture Notes in Computer Science*, pages 408–421. Springer, 1997.
12. Christine Parent and Stefano Spaccapietra. Issues and Approaches of Database Integration. *CACM*, 41(5):166–178, 1998.
13. James E. Rumbaugh. OMT: The Object Model. *Journal of Object-Oriented Programming*, 7(9):21–27, February 1995.

Object-Oriented Database as a Dynamic System with Implicit State

Kazem Lellahi[1] and Alexandre Zamulin[2]*

[1] LIPN, UPRESA 7030 C.N.R.S
Université de Paris 13, Institut Galilée
99, Av. J.B. Clément, 93430 Villetaneuse France
`kl@lipn.univ-paris13.fr`, fax +33 (0)1 4826 0712
[2] A.P. Ershov Institute of Informatics Systems
Siberian Division of Russian Academy of Sciences
Novosibirsk 630090, Russia
`zam@iis.nsk.su`, fax: +7 3832 323494

Abstract. A formalization of object-oriented database concepts in the context of algebraic specifications with implicit state is proposed. An object database schema is represented as a dynamic system and an object database instance as a state algebra. The paper also provides a formalization of binding modes and a rigorous treatment of null value.

Keywords: object modeling, object-oriented database, dynamic system, implicit state, state update.

1 Introduction

Object-oriented systems deal with collections of objects. An object has a state and a behavior. The behavior of an object serves to access or update its state. Objects are organized in an *inheritance* hierarchy. Another important aspect of object modeling is *overloading*, which is the possibility of giving the same name to several attributes or methods.

The need for formal definition of these and other pertinent object concepts is widely recognized. Papers and books devoted to this problem include database approaches [2,15,5,13,14], conventional algebraic approaches [2,9], and model-based approaches [6,22]. The database approaches have resulted in the proposal of the `ODMG` object model and the object definition language `ODL` and object query language `OQL` based upon it [4,5]. However, the model is not formalized.

A formal object model `FOR` (`F`unctional-`O`bject-`R`elational) has been introduced in [17] and further developed in [24,16]. The main characteristic of the model is its similarity with relational model and relational algebra, that is, a clear separation between database schema, database instance and query. However, that model does not deal with an important aspect of object databases, namely updating methods. This paper stresses on updating aspects.

* The work of the second author is supported in part by Russian Foundation for Basic Research under Grant 01-01-00787.

Another interesting approach is adopted in collection types [3,8,7]. The main idea of this approach is to enrich pure functional programming techniques with facilities for manipulating collections of data. It does not, however, address the problem of state transformation and does not introduce any concept of object. Moreover, since only total functions are allowed it this approach, it does not clearly treat null values that are quite typical in practical database applications.

Thus, no formal object-oriented data model with the same authority as the relational model and no object algebra so elegant as the relational algebra has yet emerged. In all likelihood, this is explained by the fact that all previous formalization attempts were mainly based on conventional algebraic specification technique. However, as it is noted in [21], the algebraic framework so far has been inadequate in describing the dynamic properties of objects and their state transformations as well as more complex notions typical of object-oriented paradigm such as object identity. A way for modeling these features is proposed in some approaches based on the notion of implicit state [12,10,11,25]. This technique seems to be very convenient for representing updating methods. Partial functions used in conventional algebraic specification approaches (for example, in the specification language CASL [19]) seem to be a convenient way for dealing with null values. We are going to use these techniques to solve both problems.

In general, the present paper proposes a new formalism for describing the main aspects of objects represented in the ODMG object model [4]. We structure objects into classes within a hierarchy with a formal definition of schema. However, our definition of schema takes into account all static and all dynamic aspects of objects as well as the concepts of inheritance, object identity and overloading. Attributes and methods are treated semantically in a uniform way as partial functions. This allows us to give a uniform rigorous formalization of null values, undefinedness, and binding modes. This also allows us to define a database schema in a way similar to an algebraic specification of data types and to view a database instance as a state algebra of that specification. Basic operations updating the state are formally defined by what we call an *applicable update set*. Such a set consists of a set of parallel actions operating on a valid state and providing a valid state. Each action creates or destroys an object or updates an attribute. In a database context an applicable update set can be regarded as a *transaction*.

The rest of the paper is organized as follows. In Sect. 2 we present our formal object model. Section 3 is the core of the paper. We define there a database state as a state algebra and we present a new formalism for describing state updates. In Sect. 4 we define elementary expressions of the object model, and we draw some conclusions and indicate some further work in Sect. 5.

2 The Object Data Model

The Type System: The model is based on a type system with the following grammar:

$$\mathcal{T} ::= \texttt{BASE} \mid \texttt{CLASS} \mid \texttt{rec IDENT} : \mathcal{T}, ..., \texttt{IDENT} : \mathcal{T} \texttt{ end} \mid \texttt{set } \mathcal{T} \qquad (1)$$

where BASE and CLASS are two disjoint non empty sets of names representing *basic types* and *class names*, respectively, IDENT is a set of identifiers, and rec and set are two type constructors serving for the creation of record types and set types, respectively. Elements of $\mathcal{T}$ are *type expressions* (or *types*). We say that the type expression t *contains* the class name c if one of the following holds:

- t is c,
- t is rec $p_1 : t_1, ..., p_n : t_n$ end and at least one of t_i contains c,
- t is *set* t_1 and t_1 contains c.

In the sequel, $\mathcal{T}^*$ stands for the set of all sequences of elements of $\mathcal{T}$ and $\mathcal{T}^u$ for $\mathcal{T} \cup \{\texttt{void}\}$ where void is a special type whose semantics will be explained later. It is assumed that BASE and CLASS are endowed with a syntactic equality '='. This equality is extended to $\mathcal{T}$, $\mathcal{T}^*$ and $\mathcal{T}^u$ in an obvious way. An element r of $\mathcal{T}^*$ is either the empty sequence or has the form $t_1 \cdots t_n$, where $n > 0$ and $t_i \in \mathcal{T}$ for all i $(1 \leq i \leq n)$.

The Database Schema: Roughly speaking, an object database schema is a set of class definitions. However, providing a formal definition of the object schema is not an obvious task. This is mainly caused by interference between inheritance hierarchy and overloading facilities. Let us have two enumerable non-empty disjoint sets ATT (attribute names) and METH (method names) having no common elements with BASE and CLASS.
An *object database schema* (or *schema* in short) $\mathcal{S}$ then consists of:

- a finite non-empty subset C of CLASS,
- a binary acyclic relation *isa* over C, and
- three partial functions $att : C \times \texttt{ATT} \to \mathcal{T}$, $con : C \times \mathcal{T}^* \to \{\texttt{void}\}$ and $meth : C \times \texttt{METH} \times \mathcal{T}^* \to \mathcal{T}^u$

such that:

- for every $c \in C$ at least one of the three sets, $isa(c) = \{c' \mid c \; isa \; c'\}$, $att(c) = \{(a,t) \mid att(c,a) = t\}$ and $meth(c) = \{(m,r,t) | \; meth(c,m,r) = t\}$, is not empty; and
- for every $c \in C$, if the set $con(c) = \{r | \; con(c,r) = \texttt{void}\}$ is not empty, then $att(c)$ is not empty, too; and
- for any type t occurring in the domain or in the range of functions *att*, *con* or *meth* and containing a class name c, $c \in C$.

The relation *isa* defines a *hierarchy* over C. We say c *inherits* c' whenever $c \; isa \; c'$. For each class name c in C, the tuple $(c, isa(c), att(c), con(c), meth(c))$ is called a *class definition* in $\mathcal{S}$ with name c. A class definition thus has a unique name. A pair (a,t) in $att(c)$ is called the declaration of an *attribute* of c with *name* a, and *type* t. Similarly, an r in $con(c)$ is called the declaration of a *constructor* of c with *rank* r, and a triple (m,r,t) in $meth(c)$ is called the declaration of a *method* of c with *name* m, *rank* r, and *type* t. The pair (r,t) is the *profile* of the method. Informally, constructors serve for object initialization and methods represent parameterized computations or updates over objects. The result type of the latter is `void`.

Some remarks explaining the above definition are in order at this point.
1) The functionality of *att* and *meth* indicates: *attribute overloading is not allowed within a class* while *method overloading within a class is allowed provided that they do not have the same rank.* Therefore, an attribute (a, t) and a method (m, r, t) in a class c should be respectively regarded as (c, a, t) or (c, m, r, t) in the whole schema. However, we identify an attribute by its name where no confusion is possible.
2) The first condition of the definition requires that *a class has at least one attribute or one method, otherwise it inherits at least one class.*
3) The second condition requires that *a class has at least one attribute if it has a constructor.*
4) The last condition of the definition requires that *an attribute or a method does not refer to a class name outside of the schema.*

The Subtyping Relation: Each schema generates in a natural way a partial order over types. Indeed, the acyclicity of *isa* implies that its transitive and reflexive closure is a partial order over C. This partial order is called the *inheritance relation* and is denoted by $\leq_{isa}$. We read $c <_{isa} c'$ as *c is a subclass of* c' or c' *is a superclass of* c. The partial order $<_{isa}$ is extended to a partial order $\preceq_{\mathcal{T}}$ over $\mathcal{T}$ in the following way:

– if b is a basic type then $b \preceq_{\mathcal{T}} b$, and if $c \leq_{isa} c'$ then $c \preceq_{\mathcal{T}} c'$;
– if $t \preceq_{\mathcal{T}} t'$ then $(set\ t) \preceq_{\mathcal{T}} (set\ t')$;
– if $t_1 \preceq_{\mathcal{T}} t'_1, ..., t_k \preceq_{\mathcal{T}} t'_k$ and $k \leq n$, then
  $\texttt{rec}\ p_1 : t_1, ..., p_n : t_n\ \texttt{end} \preceq_{\mathcal{T}} \texttt{rec}\ p_1 : t'_1, ..., p_k : t'_k\ \texttt{end}$.

We read $t \preceq_{\mathcal{T}} t'$ as *t is a subtype of* t' or t' *is a supertype of* t. The partial order $\preceq_{\mathcal{T}}$ is extended to a partial order $\preceq^*_{\mathcal{T}}$ on $\mathcal{T}^*$ and to a partial order $\preceq^u_{\mathcal{T}}$ on $\mathcal{T}^u$ by setting $\texttt{void} \preceq^u_{\mathcal{T}} \texttt{void}$ and $t_1 \cdots t_n \preceq^*_{\mathcal{T}} t'_1 \cdots t'_k$ if either both sequences are empty or $k = n$ and $t_i \preceq^*_{\mathcal{T}} t'_i$ for all i, $1 \leq i \leq n$. In the sequel, we refer to any of these ordering relations as a *subtype relation*, and we omit the subscripts where they are understood from the context.

Hierarchical Consistency (Overloading and Overriding Problems): Roughly speaking, overriding consists of hiding an attribute or a method of a class in a subclass by redefining it. More precisely,

> we say that an attribute (a, t') of a class c' is *overridden* in a class c if $c <_{isa} c'$ and there is an attribute (a, t) in c. Similarly, a method $m : r' \to t'$ of a class c' is said to be overridden in a class c if $c <_{isa} c'$ and there is a method $m : r \to t$ in c such that $r' \preceq^*_{\mathcal{T}} r$.

This overriding concept is similar, but more general, to that of programming languages `JAVA` [20] and `C++` [23]. In these languages overriding is strong in the sense that the two methods, in the class and in the subclass, must have the same profile. We allow the domain of an overriding method to be a supertype of the domain of the overidden method. This phenomena is known under the name *contravariance* of parameter types (see [1] for details). The problem of overriding is a delicate problem in the object-oriented paradigm since overridden methods can cause inconsistency during binding, and our definition of schema can give way to such inconsistency. To avoid it, some conditions must be satisfied.

Definition 1 (covariance) The database schema $\mathcal{S}$ is said to satisfy the covariance condition iff

1. for every attribute (a, t') in a class c', if it is overridden in a class c as (a, t) then $t = t'$, and
2. for every method (m, r', t') in a class c', if it is overridden in a class c as (m, r, t) then $t \preceq^u_{\mathcal{T}} t'$. ■

Thus, an attribute type cannot be changed in a subclass, but the type of a method can be changed to its subtype. Attribute overriding as it is defined is just a possibility to define the same attribute in a subclass, which can be useful for avoiding clashes in case of multiple inheritance. Covariance can be checked automatically.

Schema Closure: It can be easily seen that the model supports multiple inheritance. Indeed, apart of acyclicity, there is no other restriction on *isa*. Thus, an attribute name (a method name) can occur in several superclasses of a class c with the same or different type (profile). The problem is which of these attributes (methods) must be inherited in c ? The answer is the attribute (method) that is in the lower superclass of c provided that such a lower superclass exists. Therefore, we need another condition: the *minimal condition.* Formally, we define
$super^*(c, a) = \{c' \mid c \leq_{isa} c' \ \wedge \ (\exists t \in \mathcal{T} \ \ att(c', a) = t)\}$, and
$super^*(c, m, r) = \{c' \mid c \leq_{isa} c' \ \wedge \ (\exists t \in \mathcal{T}^u \ \ meth(c', m, r) = t)\}$.
An element of $super^*(c, a)$ is either c or a superclass of c in which a is the name of an attribute. An element of $super^*(c, m, r)$ is either c or a superclass of c in which m is the name of a method with parameter types r.

Definition 2 (minimal condition) We say that a schema $\mathcal{S}$ *satisfies the minimal condition* if the following holds:

– For every class name c, attribute name a, method name m and sequence of types r occurring in $\mathcal{S}$, each of two sets $super^*(c, a)$ and $super^*(c, m, r)$ has a minimum whenever it is not empty.

These minimums are denoted by $ResA(c, a)$ and $ResM(c, m, r)$ and called the *resolution of a in c* and the *resolution of m in c with respect to r*, respectively. ■

Minimal conditions can be checked automatically.
If $ResA(c, a) = c'$, then there is an attribute (a, t) in c' and $c \leq_{\mathcal{T}_C} c'$. If $c' = c$, then (a, t) is explicitly defined in c, that is $att(c, a) = t$. Otherwise, $att(c, a)$ is not defined explicitly, but (a, t) may be treated as an implicit (inherited) attribute

of c. Thus, we can extend att to (c, a) by defining $\overline{att}(c, a) = t$. In a similar way, if $ResM(c, m, r) = c'$, then there is a method $m : r \to t'$ in c'. Then, either $c = c'$ and $m : r \to t'$ is an explicit method in c, that is $meth(c, m, r) = t'$, or $m : r \to t'$ is an inherited method in c and we can define $\overline{meth}(c, m, r) = t'$. Therefore, when the schema satisfies the minimum condition, the following rules extend the functions att and $meth$ to all inherited attributes and methods:

$$\frac{ResA(a, c) = c' \quad att(c', a) = t}{\overline{att}(c, a) = t} \qquad \frac{ResM(m, c, r) = c' \quad meth(c', m, r) = t'}{\overline{meth}(c, m, r) = t'} \tag{2}$$

It is not difficult to see that $\overline{\mathcal{S}} = (C, isa, \overline{att}, con, \overline{meth})$ is a database schema, which we call the *closure* of $\mathcal{S}$ *under inheritance*. Indeed, the schema $\overline{\mathcal{S}}$ is obtained from $\mathcal{S}$ by adding all inherited attributes and inherited methods, using rules 2. According to these rules, $\overline{att}$ and $\overline{meth}$ are partial functions and no new class name is added to $\mathcal{S}$. Moreover, no new parameter type can be created, therefore $\overline{(\overline{\mathcal{S}})} = \overline{\mathcal{S}}$. Note that constructors are not inherited.

The above discussion shows that a well-defined schema should satisfy covariance and minimality conditions. Such a schema is called *hierarchically consistent.* Hierarchical consistency guarantees that inheriting and binding will not cause any ambiguities.

Theorem 1 ([16]) If $\mathcal{S}$ is a hierarchically consistent schema, then so is $\overline{\mathcal{S}}$.

We consider only a hierarchically consistent schema in the sequel.

3 Algebra and Database

Basic Algebra: A *basic algebra* $\mathtt{B}$ associates with each type t a set $\mathtt{B_t}$, its *carrier*, and with each operation of t a partial function, its *implementation.* The carriers of our types are defined recursively as follows:

- The carrier of each basic type t is an enumerable set $\mathtt{B_t}$. The carriers of basic types are assumed to be pairwise disjoint.
- The carrier of each $c \in C$ is the set `Oid`, which is supposed to be a special set disjoint from all basic types carriers.
- The carrier of *set* t is $\mathcal{P}_{\mathtt{f}}(\mathtt{B_t})$, the set of finite subsets of $\mathtt{B_t}$.
- The carrier of $\mathtt{rec}\ p_1 : t_1, ..., p_n : t_n\ \mathtt{end}$ is the set of tuples $(\mathtt{v_1}, \ldots, \mathtt{v_n})$ so that at least one of $\mathtt{v_i}$ is in $\mathtt{B_i}$ and any other $\mathtt{v_j}$ can be $\perp$ where $\perp$ is a special value that is neither an element of a basic type nor an element of `Oid`.

Values of basic types are called *observable values*, those of `Oid` *non-observable values* or *object identities.* The special value $\perp$ is called the *null value* denoted also by `NULL` in the sequel.

We assume that each basic type is endowed with some operations, and each of them is implemented as a partial function $\mathtt{op^A} : \mathtt{B_{t_1}} \times \cdots \times \mathtt{B_{t_n}} \to \mathtt{B_t}$ when $n > 0$, otherwise $\mathtt{op^B} \in \mathtt{B_t}$. Each set type is assumed to be endowed with usual set operations. A record type $r = \mathtt{rec}\ p_1 : t_1, ..., p_n : t_n\ \mathtt{end}$ possesses a number

of partial projection operations $p_i : r \to t_i$ and a record construction operation
$rec : t_1, ..., t_n \to r$
mapped in $\mathtt{B}$ to a partial function
$\mathtt{rec}^{\mathtt{B}} : \mathtt{B}'_{\mathtt{t}_1} \times ... \times \mathtt{B}'_{\mathtt{t}_n} \to \mathtt{B}_{\mathtt{r}}$,
where $\mathtt{B}'_{\mathtt{t}_i} = \mathtt{B}_{\mathtt{t}_i} \cup \{\perp\}$, so that $\mathtt{p}^{\mathtt{B}}_{\mathtt{i}}(\mathtt{rec}^{\mathtt{B}}(\mathtt{v}_1, ..., \mathtt{v}_n)) = \mathtt{v}_i$ if $\mathtt{v}_i$ is not $\perp$ and it is undefined otherwise. Note that the result of a projection function is undefined rather than NULL if a record is projected to a field with value NULL. This helps us to avoid typing problems which are inevitable if NULL is used in expressions.

The equality operation for two records, $\mathtt{r}_1$ and $\mathtt{r}_2$, of the same type $r =$ rec $p_1 : t_1, ..., p_n : t_n$ end is defined as follows: $\mathtt{r}_1 = \mathtt{r}_2$ iff, for each $i = 1, \cdots, n$, either both $\mathtt{p}_i(\mathtt{r}_1)$ and $\mathtt{p}_i(\mathtt{r}_2)$ are defined and $\mathtt{p}_i(\mathtt{r}_1) = \mathtt{p}_i(\mathtt{r}_2)$ or both $\mathtt{p}_i(\mathtt{r}_1)$ and $\mathtt{p}_i(\mathtt{r}_2)$ are not defined.

The only predefined operation of $c \in C$ is the comparison operation "=", such that $\mathtt{o} = \mathtt{o}'$ is true iff both $\mathtt{o}$ and $\mathtt{o}'$ are the same element of Oid.

State Algebra: A state algebra represents a database state. Let $\mathtt{B}$ be a basic algebra, $\mathcal{S} = (C, isa, att, con, meth)$ a database schema, and $\overline{\mathcal{S}}$ its closure under inheritance.
A *state algebra* $\mathtt{A}$, over $\mathcal{S}$ is created in the following way:

1. A finite partial identity function, $\mathtt{id}^{\mathtt{A}}_{\mathtt{cc}} : \mathtt{Oid} \to \mathtt{Oid}$, is associated with each $c \in C$ so that $id_{cc}(o)$ and $id_{c'c'}(o)$ are both defined iff there is c'' such that $c'' \leq_{isa} c$ and $c'' \leq_{isa} c'$ and $id_{c''c''}(o)$ is defined.
   We denote by $\mathtt{A}_{\mathtt{c}}$ the range of the function $\mathtt{id}^{\mathtt{A}}_{\mathtt{cc}}$. For a basic type t, we define $\mathtt{A}_{\mathtt{t}} = \mathtt{B}_{\mathtt{t}}$ and we then expand $\mathtt{A}_{\mathtt{t}}$ to any type t created with the use of the type constructors.
2. A partial function $\mathtt{a}^{\mathtt{A}}_{\mathtt{ct}} : \mathtt{A}_{\mathtt{c}} \to \mathtt{A}_{\mathtt{t}}$ is associated with each attribute $(a, t) \in att(c)$ so that, for each pair (c, c'), if $c <_{isa} c'$ and $\mathtt{o} \in \mathtt{A}_{\mathtt{c}}$, then $\mathtt{a}^{\mathtt{A}}_{\mathtt{ct}}(\mathtt{o}) = \mathtt{a}^{\mathtt{A}}_{\mathtt{c't}}(\mathtt{o})$. Such a function is called an *attribute function.*

Thus, each subclass inherits part of its superclass attribute functions. In the sequel, the set $\mathtt{A}_{\mathtt{c}}$ is called the *extent* of c in the state $\mathtt{A}$. Each $\mathtt{o}$ in $\mathtt{A}_{\mathtt{c}}$ is called an *object identity of the class c.* Note that the set of object identities of a superclass includes object identities of its subclasses. Indeed, if c is a subclass of c' then in clause 1 above c'' is c. Thus, if $id_{cc}(o)$ is defined so is $id_{c'c'}(o)$, in other words $\mathtt{A}_{\mathtt{c}} \subseteq \mathtt{A}_{\mathtt{c}'}$. Therefore, *the semantics of inheritance in a state algebra is set inclusion.*

If $\mathtt{o} \in \mathtt{A}_{\mathtt{c}}$, then $\mathtt{a}^{\mathtt{A}}_{\mathtt{ct}}(\mathtt{o})$ is an *attribute* of $\mathtt{o}$. An *object* is a pair (o, obs) where o is an object identity and obs is the tuple of its attributes called *object's state.* We write sometimes "object o" meaning the object with the identity o. If $\mathtt{id}^{\mathtt{A}}_{\mathtt{cc}}(\mathtt{o})$ is defined, we say that c is a type of $\mathtt{o}$. Furthermore, if there is no $c' <_{isa} c$ such that $\mathtt{id}^{\mathtt{A}}_{\mathtt{c'c'}}(\mathtt{o})$ is defined, we say that c is the *most specific type* of o. An object o is a *proper object* of a class c iff it is not in any subclasses of c. In practice, some classes are generic and may not have proper objects (because either no attribute is defined or inherited in the class or some its methods are not implemented). Such a class is called an *interface* in [4]. For simplicity, we do not make difference between classes and interfaces in this paper.

Several state algebras over a database schema $\mathcal{S}$ can have the same base algebra. Following the notation of [10], we denote the set of all state algebras, with the same base $\mathtt{B}$, by $\mathtt{state}_{\mathtt{B}}(\mathcal{S})$ and mean by a $\mathcal{S}_{\mathtt{B}}$-state a state over $\mathcal{S}$ with the basic algebra $\mathtt{B}$.

We say that a value $\mathtt{v}$ of type t contains a value (object identity) $\mathtt{o}$ of class c in the state $\mathtt{A}$ if one of the following holds:

- t is c and $\mathtt{v}$ is $\mathtt{o}$,
- t is $\mathtt{rec}\ p_1 : t_1, ..., p_n : t_n\ \mathtt{end}$ and at least one of $\mathtt{p}_\mathtt{i}^\mathtt{A}(\mathtt{v})$ contains $\mathtt{o}$,
- t is *set* t_1 and at least one $\mathtt{w} \in \mathtt{v}$ contains $\mathtt{o}$.

To define the interpretation of method names, we firstly need to introduce a formal notion of *state update*.

State Update: One state can be transformed into another by a *state update*.

Definition 3 A *state update* in a $\mathcal{S}_B$-state $\mathtt{A}$ is a triple $(\mathtt{f}_{\mathtt{ct}}, \mathtt{o}, \mathtt{v})$ where $\mathtt{f}_{\mathtt{ct}}$ is either an attribute symbol of type t in class c or the identity function symbol $\mathtt{id}_{\mathtt{cc}}$, and the other two elements are the following:

- for an update $(\mathtt{id}_{\mathtt{cc}}, \mathtt{o}, \mathtt{o})$, the object identity $\mathtt{o}$ is not in $\mathtt{A}_{cl}$, for any $cl \in C$,
- $\mathtt{o} \in \mathtt{A}_\mathtt{c}$ and $\mathtt{v}$ is either an element of $\mathtt{A}_\mathtt{t}$ or the symbol $\perp$ in all other cases. ■

Note that $\mathtt{f}_{\mathtt{ct}}$ in a triple $(\mathtt{f}_{\mathtt{ct}}, \mathtt{o}, \mathtt{a})$ is actually a function symbol, i.e., a function name qualified with its profile, which helps us to avoid ambiguity when attribute names are overloaded in different classes.

A state update $\alpha = (\mathtt{f}_{\mathtt{ct}}, \mathtt{o}, \mathtt{a})$ serves for the transformation of a $\mathcal{S}_B$-state $\mathtt{A}$ into a new algebra $\mathtt{A}\alpha$ in the following way:

- $\mathtt{g}^{\mathtt{A}\alpha}_{\mathtt{c't'}}$ is the same as $\mathtt{g}^{\mathtt{A}}_{\mathtt{c't'}}$ for any $\mathtt{g}_{\mathtt{c't'}}$ different from $\mathtt{f}_{\mathtt{ct}}$;
- $\mathtt{f}^{\mathtt{A}\alpha}_{\mathtt{ct}}(\mathtt{o}) = \mathtt{a}$ if $\mathtt{a}$ is not $\perp$, $\mathtt{f}^{\mathtt{A}\alpha}_{\mathtt{ct}}(\mathtt{o})$ becomes undefined otherwise;
- $\mathtt{f}^{\mathtt{A}\alpha}_{\mathtt{ct}}(\mathtt{o'}) = \mathtt{f}^{\mathtt{A}}_{\mathtt{ct}}(\mathtt{o'})$ for any $\mathtt{o'}$ different from $\mathtt{o}$.

Following Gurevich [12], we say that $\mathtt{A}\alpha$ is obtained by firing the update α on the state $\mathtt{A}$. Roughly speaking, firing a state update either inserts an element into the definition domain of an attribute function $\mathtt{a}_{\mathtt{ct}}$ or modifies the value of such a function at one point in its definition domain or removes an element from the definition domain. The state update $(\mathtt{id}_{\mathtt{cc}}, \mathtt{o}, \mathtt{o})$ in fact extends the set of object identities of a class c by a new element $\mathtt{o}$ and the state update $(\mathtt{id}_{\mathtt{cc}}, \mathtt{o}, \perp)$ contracts the set of object identities of a class c.

Definition 4 A set Γ of state updates is *inconsistent* if it contains

- two state updates $\alpha_1 = (\mathtt{f}_{\mathtt{ct}}, \mathtt{o}, \mathtt{v})$ and $\alpha_2 = (\mathtt{f}_{\mathtt{ct}}, \mathtt{o}, \mathtt{v'})$ s.t. $\mathtt{v} \neq \mathtt{v'}$ (two state updates defining an attribute function differently at the same point), or
- an $\alpha_1 = (\mathtt{a}_{\mathtt{ct}}, \mathtt{o}, \mathtt{v})$ and $\alpha_2 = (\mathtt{id}_{\mathtt{c'c'}}, \mathtt{o'}, \perp)$ such that either c' is c and $\mathtt{o} = \mathtt{o'}$ or t contains c' and $\mathtt{v}$ contains $\mathtt{o'}$ (an object is removed from a class extent while an attribute function is forced to use it);

the update set is *consistent* otherwise. ■

A consistent update-set Γ applied to a $\mathcal{S}_{\mathtt{B}}$-state $\mathtt{A}$ transforms it into an algebra $\mathtt{A}'$ by simultaneous firing all $\alpha \in \Gamma$. If Γ is inconsistent, $\mathtt{A}'$ is not defined. If Γ is empty, $\mathtt{A}'$ is the same as $\mathtt{A}$. Following [18], we denote the application of Γ to a state $\mathtt{A}$ by $\mathtt{A}\Gamma$.

Definition 5 The *sequential union* of two consistent update-sets Γ_1 and Γ_2, denoted by $\Gamma_1;\Gamma_2$, is a consistent update set created as follows:

– delete from $\Gamma_1 \cup \Gamma_2$ any $\alpha_1 \in \Gamma_1$ for which there is an $\alpha_2 \in \Gamma_2$, such that $\{\alpha_1, \alpha_2\}$ is inconsistent, and any $\alpha_2 = (\mathtt{id}_{\mathtt{cc}}, \mathtt{o}, \perp) \in \Gamma_2$ if $\alpha_1 = (\mathtt{id}_{\mathtt{cc}}, \mathtt{o}, \mathtt{o}) \in \Gamma_1$. ■

The above definition of a consistent update set permits us to check whether there are internal contradictions in an update set. However, a consistent update set can contradict the state to which it should be applied. Therefore, a notion of an applicable update set is needed.

Definition 6 A consistent update set Γ is *applicable* to a state $\mathtt{A}$ if either Γ does not contain an update of the form $(\mathtt{id}_{\mathtt{cc}}, \mathtt{o}, \perp)$, or for each $(\mathtt{id}_{\mathtt{cc}}, \mathtt{o}, \perp) \in \Gamma$:

– either there is no $\mathtt{a}^{\mathtt{A}}_{\mathtt{c't}}$ such that, given an $\mathtt{o}' \in \mathtt{A}_{\mathtt{c'}}$, the attribute $\mathtt{a}^{\mathtt{A}}_{\mathtt{c't}}(\mathtt{o}')$ contains $\mathtt{o}$ or $(\mathtt{a}_{\mathtt{c't}}, \mathtt{o}', \perp) \in \Gamma$ (either the object to be deleted not referenced by any other object or all such references are deleted in this update set) ;
– either $\mathtt{a}^{\mathtt{A}}_{\mathtt{ct}}(\mathtt{o})$ is not defined for any $(a, t) \in att(c)$ or $(\mathtt{a}_{\mathtt{ct}}, \mathtt{o}, \perp) \in \Gamma$ (either all attributes of this object are not defined in $\mathtt{A}$ or they are made undefined in this update set). ■

According to the above definition, we do not allow an automatic cascaded deletion of objects. If such a deletion is needed, it should be programmed in a method. Unfortunately, it is not sufficient to have an applicable update set to produce a $\mathcal{S}_{\mathtt{B}}$-state. The problem is caused by inheritance. For example, firing an $\alpha = (\mathtt{id}_{\mathtt{cc}}, \mathtt{o}, \mathtt{o})$ in an $\mathcal{S}_{\mathtt{B}}$-state $\mathtt{A}$ updates $\mathtt{id}^{\mathtt{A}}_{\mathtt{cc}}$ but does not change $\mathtt{id}^{\mathtt{A}}_{\mathtt{c'c'}}$ for all c', $c \leq_{isa} c'$. Thus, to guarantee that a state transformation produces a state algebra, we define a closure of an update set.

Definition 7 For a consistent update set Γ, the *closure* $\overline{\Gamma}$ of Γ is constructed as follows:

– for each $(\mathtt{id}_{\mathtt{cc}}, \mathtt{o}, \mathtt{o}) \in \Gamma$, insert $(\mathtt{id}_{\mathtt{c'c'}}, \mathtt{o}, \mathtt{o})$ in $\overline{\Gamma}$ for all c' such that $c \leq_{isa} c'$;
– for each $(\mathtt{id}_{\mathtt{cc}}, \mathtt{o}, \perp) \in \Gamma$ having c'' as the most specific type of $\mathtt{o}$, insert $(\mathtt{id}_{\mathtt{c'c'}}, \mathtt{o}, \perp)$ in $\overline{\Gamma}$ for all c' such that $c'' \leq_{isa} c'$.
– for each $(\mathtt{a}_{\mathtt{ct}}, \mathtt{o}, \mathtt{v}) \in \Gamma$ having c'' as the most specific type of $\mathtt{o}$, insert $(\mathtt{a}_{\mathtt{c't}}, \mathtt{o}, \mathtt{v})$ in $\overline{\Gamma}$ for all c' such that $c'' \leq_{isa} c'$ and $(a, t) \in att(c')$. ■

The first clause of the above definition guarantees that inserting an object in a class we also insert it into all superclasses of this class. The second clause guarantees that deleting an object from a class we also delete it from all subclasses in which it exists. The third clause guarantees that an update of an attribute function for an object of a certain class will affect the corresponding attribute function of all classes that are supertypes of the most specific type of the object.

Fact 1 The closure $\overline{\Gamma}$ of a consistent update set Γ is well defined and is consistent. Moreover, if Γ is applicable to a state $\mathtt{A}$ so is $\overline{\Gamma}$. ■

We say a consistent update set Γ is closed if $\overline{\Gamma} = \Gamma$.

Fact 2 1) for any state algebra $\mathtt{A}$ and any closed applicable update set Γ, $\mathtt{A}\Gamma$ is a state algebra.
2) If Γ_1 and Γ_2 are closed applicable update sets, so is $\Gamma_1;\Gamma_2$.
3) For any state $\mathtt{A}$ and applicable update sets Γ_1 and Γ_2, $\mathtt{A}(\Gamma_1;\Gamma_2) = (\mathtt{A}\Gamma_1)\Gamma_2$. ■

The set of all closed applicable updates sets in a $\mathcal{S}_\mathtt{B}$-state $\mathtt{A}$ is denoted by $\overline{\mathtt{update}_\mathtt{A}}(\mathcal{S})$. It serves for the definition of the semantics of the type `void` in the sequel, i.e., $\mathtt{A}_{\mathtt{void}} = \overline{\mathtt{update}_\mathtt{A}}(\mathcal{S})$. We consider only closed consistent update sets from now on. If there is no possibility of confusion, we simply write an *update set* in the sequel.

Database: In practice, we are interested only in a part of $\mathtt{state}_\mathtt{B}(\mathcal{S})$ called *valid states.* A valid state is normally a state satisfying a set of constraints (or axioms) specified within the schema. In an object-oriented database the only *admissible* updates are those produced by an updating method. Specifying the constraints and checking the validity of a state is an important and sophisticated problem which is out of the scope of this paper. Formally, it is sufficient to assume that the application of any admissible update to a valid state produces a valid state. Thus, without loss of generality we can suppose that all states are valid.

In the definitions that follow, for any state algebra $\mathtt{A}$ and any sequence of types $r = t_1 \cdots t_n$, we denote $\mathtt{A}_{\mathtt{t}_1} \times \cdots \times \mathtt{A}_{\mathtt{t}_n}$ by $\mathtt{A}_\mathtt{r}$ which is a singleton set if $n = 0$.

Definition 8 A *database* $\mathtt{DB(B)}$ consists of:

1. A subset $|\mathtt{DB(B)}|$ of $\mathtt{state}_\mathtt{B}(\mathcal{S})$ called the *carrier* of $\mathtt{DB(B)}$,
2. for each $c \in C$, $r \in con(c)$ and $\mathtt{A} \in |\mathtt{DB(B)}|$, a partial function $\mathtt{c}^\mathtt{A}_\mathtt{cr} : \mathtt{A}_\mathtt{c} \times \mathtt{A}_\mathtt{r} \to \mathtt{A}_{\mathtt{void}}$, and
3. for each $c \in C$, $(m, r, t) \in \overline{meth}(c)$ and $\mathtt{A} \in |\mathtt{DB(B)}|$, a partial function $\mathtt{m}^\mathtt{A}_\mathtt{cr} : \mathtt{A}_\mathtt{c} \times \mathtt{A}_\mathtt{r} \to \mathtt{A}_\mathtt{t}$ such that if $c' <_{isa} c$ and $m : r \to t$ is inherited in c' from c, then $\mathtt{m}^\mathtt{A}_\mathtt{cr}(\mathtt{o}, \mathtt{v}) = \mathtt{m}^\mathtt{A}_{\mathtt{c'r}}(\mathtt{o}, \mathtt{v})$ for each $(\mathtt{o}, \mathtt{v}) \in \mathtt{A}_{\mathtt{c'}} \times \mathtt{A}_\mathtt{r}$. ■

Since constructors are not inherited, the clause 2 says that constructors in a subclass are different from constructors in its superclasses. Since an inherited and an overridden method in a class can not have the same profile, functions $\mathtt{m}^\mathtt{A}_\mathtt{cr}$ in clause 3 are safe. This clause says that if a class inherits some method from a superclass, then both superclass objects and subclass objects are supplied with the same method. If a subclass overrides a method of its superclass, its objects are supplied with a different method. In the case when a method is never overridden in a subclass, it can be associated both with the class itself and all its subclasses. The former case serves for providing dynamic (late) binding, while the last one for providing the static (early) binding which can be considered as an optimization of the former one. Note that some methods can be left unimplemented if interfaces are considered.

If the method type t is `void`, then $\mathtt{A_t}$ is the set of admissible update sets in $\mathtt{A}$, i.e., such a method produces an update set used for state transformation.

4 Elementary Expressions

We introduce several rules for creating expressions involving attribute and method names. Special kind of expressions called *transition expressions* denote transitions from one state to another. Interpretation of these expressions is done with the use of update sets. In this way, a mechanism of *message passing* is provided on one hand, and a base for a more developed data manipulation language is created on the other hand. Let $\mathcal{S} = (C, isa, att, meth)$ be a database schema, $\mathtt{DB(B)}$ a database, $\mathtt{A} \in |\mathtt{DB(B)}|$ a database state, and Y a set of C-sorted variables Then we define the set of $\mathcal{S}$-expressions, $T(\mathcal{S}, Y)$. Given a valuation function $u : Y \longrightarrow A$, we also define the interpretation $[\![\tau]\!]^A$ of an expression $\tau \in T(\mathcal{S}, Y)$.

1. If $y \in Y_c$, then y is an expression of type c, where Y_c is the c-subset of Y. **Interpretation:** $[\![y]\!]^A = u(y)$.
2. If $c' \leq c$ and τ is an expression of type c', then τ is an expression of type c. **Interpretation:** if τ evaluates in $\mathtt{A}$ to an $\mathtt{o} \in \mathtt{A_{c'}}$, then by definition $\mathtt{o} \in \mathtt{A_c}$ and can be treated as an object of type c. Thus, a subtype expression can be used in a context where an expression of its supertype is needed.
3. If (at, t) is an attribute from $\overline{att}(c)$ and τ is an expression of type c, then $\tau.at$ is an expression of type t called an *attribute access*. **Interpretation:** $[\![\tau.at]\!]^A = \mathtt{at_c^A}([\![\tau]\!]^\mathtt{A})$ if τ is defined in A and $\mathtt{at_c^A}$ is defined for $[\![\tau]\!]^A$; $[\![\tau.at]\!]^A$ is undefined otherwise. Thus, an attribute value is produced by the corresponding attribute function.
4. If τ is an expression of type t, then $\mathbf{D}(\tau)$ is an expression of type Boolean. **Interpretation:** $[\![\mathbf{D}(\tau)]\!]^\mathtt{A} = \mathtt{true}$ if τ is defined in $\mathtt{A}$, and $[\![\mathbf{D}(t)]\!]^\mathtt{A} = \mathtt{false}$ otherwise.

   This kind of expression helps us to check whether an attribute is provided with a value, i.e., whether it is not `NULL`.
5. If (m, r, t) is a method from $\overline{meth}(c)$, where $r = t_1 \ldots t_n$, and $\tau, \tau_1, \ldots, \tau_n$ are expressions of types $c, t_1, \ldots, t_n$, respectively, then $\tau.m(\tau_1, \ldots, \tau_n)$ is an expression of type t called a *method call*. The expression is called a *transition expression* if it is an expression of type `void`. The method call is always type-safe since parameter types of an overridden method are in the contravariant position with respect to the parameter types of the original method.

Interpretation: If c' is the most specific type of the object $[\![\tau]\!]^{\mathtt{A}}$ with a method (m, r', t') such that $r \leq_{isa} r'$ and $t' \leq_{isa} t$, then
$[\![\tau.m(\tau_1, \ldots, \tau_n)]\!]^{\mathtt{A}} = m^{\mathtt{A}}_{c'r'}([\![\tau]\!]^{\mathtt{A}}, [\![\tau_1]\!]^{\mathtt{A}}, \ldots, [\![\tau_n]\!]^{\mathtt{A}})$
if τ and each $\tau_i, i = 1, \ldots, n$, are defined in $\mathtt{A}$ and $m^{\mathtt{A}}_{c'r'}$ is defined for the tuple $([\![\tau]\!]^{\mathtt{A}}, [\![\tau_1]\!]^{\mathtt{A}}, \ldots, [\![\tau_n]\!]^{\mathtt{A}})$; $[\![\tau.m(\tau_1, \ldots, \tau_n)]\!]^{\mathtt{A}}$ is undefined otherwise. Thus, the interpretation of a method call in a state $\mathtt{A}$ causes the invocation of the method associated with the most specific type of the object $[\![\tau]\!]^{\mathtt{A}}$. In this way, a method name is bound exactly to a method associated with the given object independent of its current type (e.g., τ can be evaluated to an object whose the most specific type is a subtype of the type of τ).[1] This phenomena is known in the object-oriented programming paradigm as *dynamic (late) binding*. Note that the interpretation of a transition expression produces an update set used for state transformation.

6. If c is the name of a class, x a variable, and τ a transition expression using x as a term of type c and not having x as a fresh variable, then $\mathtt{new(x : c)\ in}\ \tau$ is a transition expression with a fresh variable x called *import instruction.*
Interpretation: $[\![\mathtt{new(x : c)\ in}\ \tau]\!]^{\mathtt{A}} = \overline{\Gamma}\ ;\ \overline{[\![\tau\sigma]\!]^{\mathtt{A'}}}$
where $\Gamma = \{(\mathtt{id_{cc}}, \mathtt{o}, \mathtt{o})\}$, $\mathtt{o} \in \mathtt{Oid}$ and $\mathtt{o} \notin \mathtt{A_{cl}}$ for any $cl \in C$, $\mathtt{A'} = \mathtt{A}\overline{\Gamma}$, and $\sigma = \{\mathtt{x} \mapsto \mathtt{o}\}$ is a variable assignment.
Thus, the interpretation of this transition expression causes the creation of a new object used in the subsequent transition expression.
A special case of the above expression is the object creation instruction defined as follows. If $(t_1, ..., t_n) \in con(c)$, i.e. it is a constructor in c, and $\tau_1, \ldots, \tau_n$ are expressions of types $t_1, \ldots, t_n$, respectively, then *create* $c(\tau_1, \ldots, \tau_n)$ is a transition expression called *object creation instruction.*
Interpretation: $[\![create\ c(\tau_1, \ldots, \tau_n)]\!]^{A} = \overline{\Gamma}\ ;\ \mathtt{c}^{\mathtt{A'}}(\mathtt{o}, [\![\tau_1]\!]^{\mathtt{A'}}, \ldots, [\![\tau_{\mathtt{n}}]\!]^{\mathtt{A'}})$,
where $\mathtt{A'} = \mathtt{A}\overline{\Gamma}$, $\Gamma = \{(\mathtt{id_{cc}}, \mathtt{o}, \mathtt{o})\}$, $\mathtt{o} \in \mathtt{Oid}$, and $\mathtt{o} \notin \mathtt{A_{cl}}$ for any $cl \in C$.
Thus, the interpretation of this transition expression causes the creation of a new object initialized by the corresponding constructor.
7. If τ is an expression of class c such that c is the most specific type of τ, then **delete** τ is a transition expression called a *class contraction instruction.*
Interpretation:
   - $[\![\mathbf{delete}\ \tau]\!]^{A} = \overline{\Gamma_1 \cup \Gamma_2}$, where $\Gamma_1 = \{(\mathtt{a_{ct}}, [\![\tau]\!]^{\mathtt{A}}, \perp)\}$ for all $(c, t) \in att(c)$ and $\Gamma_2 = \{(\mathtt{id_{cc}}, [\![\tau]\!]^{\mathtt{A}}, \perp)\}$ if there is in $\mathtt{A}$ no maplet $\langle \mathtt{o'} \mapsto \mathtt{v} \rangle$ such that $\mathtt{v}$ contains $[\![\tau]\!]^{A}$,.
   - $[\![\mathbf{delete}\ \tau]\!]^{A}$ in undefined otherwise.

Thus, the interpretation of this transition expression causes the deletion of an object from the database only in case it is not referenced by an attribute function of any other object; otherwise the interpretation is undefined (i.e., an object cannot safely be deleted).

[1] There might be a dilemma in choosing the right method if we have in c both (m, r, t) and (m, r', t') such that $r' \leq r$. We believe, however, that the problem can be resolved by an algorithm based on best matching of type parameters.

Fact 3 Both import instruction and object creation instruction produce applicable update sets.

Fact 4 Class contraction instruction produces an applicable update set.

5 Conclusions and Future Works

We have introduced a formal object-oriented data model and defined its semantics in the style of dynamic systems with implicit state. For this purpose, a formal algebraic model of a class of objects has been elaborated. The model naturally simulates the notions of object and object class in object-oriented database systems and object programming languages. Each object possesses a state and a unique identifier. In each database state, for each class, a finite set of unique identifiers is provided by the state algebra of the given schema. An object state is represented by a number of attributes which can be updated. Attributes are defined as functions from object identifiers to values of some other (attribute) types. An object state can be observed or updated by a method that is a state-dependent function. However, the state is an implicit argument for all methods and an implicit result for state updating methods. This has permitted us to dramatically simplify the semantics of the methods.

We have shown that the model supports all important aspects of objects, namely inheritance, overloading, overriding, and early and late binding. We have deduced a subtyping relation from inheritance and we have used it for giving a nice formalization of delicate problems such as attribute and method resolution, multiple inheritance, contravarience/covariance and binding modes.

Several particularly interesting aspects of the model have not been presented here including multiple collections, and a formal language for coding methods. We believe that our elementary term constructors, defined in Sect. 4, can be enriched with other term constructors in order to define such a formal language. We are proceeding with research in these directions.

Although we have stressed database aspects in the paper, we claim that with a slight change of terminology the present work can be seen as a formal description of object programs in programming languages too. For instance, our concept of applicable update set has allowed us to give a nice semantics of the type `void` used in most object-oriented languages. We claim that our work provides for a better understanding of the concepts of objects and for clarifying the border between object databases and object programming languages, which can help to put together ideas from the two communities.

References

1. M. Abadi and L. Cardelli. *A Theory of Objects*. Springer-Verlag, 1996.
2. C.Beeri. A Formal Approach to Object-Oriented Databases. *Database & Knowledge Engeeniring* (5), 1990, pp.353-382.
3. P. Buneman and S. Naqvi and V. Tannen and Limsoon Wong. Principle of Programming with Complex Objects and Collection types. *Theorical computer Science*, 149:3-48, 1995.

4. R. Cattel et al. *The Object Data Standard: ODMG 3.0.* Morgan Kaufmann, 2000.
5. S. Cluet. Designing OQL: Allowing object to be queried. *Information Systems* 23(5), pages 279-305, 1998.
6. S. Cheri, R. Marthey-Chirema. A model and Language for Archive DOOD Systems. *Proceedings of 2d international East-West Database Workshop, Klagenfurt, Austria, 25-28 Sept. 1994, Springer-Verlag 1995*, pp. 3-16.
7. L. Fegaras. Query Unnesting in Object-Oriented Databases. *Proceedings of the ACM SIGMOD International Conference on Managmenent of data, Seattle, Washington*, pages 49-60, 1998.
8. L. Fegaras and D. Maier. Towards an Effective Calculus for Object query languages. *In Proc. of ACM SIGMOD International Conference on Management of Data*, pages 47-58,1995.
9. J.A. Goguen and R. Diaconescu. Towards an algebraic Semantics for the object paradigm. *Recent Trends in Data Type specification*, LNCS vol. 785, 1994, pp. 1-29.
10. M.-C. Gaudel, C. Khoury, A. Zamulin. Dynamic systems with implicit state. *Fundamental Approaches to Software Engineering*, LNCS, vol. 1577, 1999, pp.114-128.
11. M. Grosse-Rhode. Algebra Transformation Systems And Their Composition. *Fundamental Approaches to Software Engineering*, LNCS, vol. 1382, pp. 107-122, 1998.
12. Y. Gurevich, Evolving Algebras 1993: Lipary Guide, Specification and Validation Methods,*Oxford University Press*, 1994.
13. W. Kim. *Modern Database Systems. The Object Model, Interoperability, and Beyond*, Addison-Wesley, 1995.
14. G. Lausen and G. Vossen. *Models and Languages of Object-Oriented Databases*, Addison-Wesley, 1997.
15. C. Lecluse and P. Richard and V. Velez. The O2 Data Model, *In François Bancilhon, Claude Delobel, and Paris Kanellakis, editors, Building an Object-Oriented Database System, The Story of O2 Morgan Kaufmann*, 1992.
16. K. Lellahi. Modeling data and objects: An algebraic viewpoint. *Proceeding of the first summer school in theoretical aspects of computer science*, Tehran-Iran July 2000 (to appear in LNCS).
17. K. Lellahi and R. Souah and N. Spyratos. An algebraic query language for Object-Oriented data Models. *DEXA97, LNCS N0 1308*, pages 519-528, 1997.
18. K. Lellahi, A. Zamulin. Dynamic Systems Based On Update Sets. Proc. of the 2nd International Conf. Computer Science and Information Technologies (CSIT'99), August 17–22, 1999, Yerevan, Armenia, 1999, pp. 346-349, also in *Reserch report No 99-03, LIPN*, Univ. Paris 13 (France), 1999.
19. P. Mosses. CASL: a guided tour of its design. *Recent Trends in Algebraic Development Techniques: Selected Papers from WADT'98*, Lisbon, Springer LNCS, vol. 1589, 1999.
20. P. Niemeyer and J. Peck. *Exploring Java. O'Reilly & Associates Inc*, 1996.
21. F.Parisi-Presicce and A. Pierantonio. Dynamic Behaviour of Object Systems. *Recent Trends in Data Type specification*, LNCS vol. 906, 1995, pp. 406-419.
22. Klause-Dieter Schewe. *Specification of Data Intensive Application Systems.* Habilitationsschrift, Technischen Universitaet Cottbus, 1995.
23. B. Stroustrup. *The* C++ *Programming Language, Third Edition.* Addison-Wesley, 1996.
24. R. Souah. *Une Sémantique Algébrique pour Bases de données Orientées objet.* PhD thesis, Université Paris-sud (Orsay), 1999.
25. E. Zucca, Fundamental Study from static to dynamic abstract data-type: an instutition transformation, *Theorical Computer science* 216 (1999), pp 109-157.

The Use of Aggregate and Z Formal Methods for Specification and Analysis of Distributed Systems

Henrikas Pranevicius

Business Informatics Department, Kaunas University of Technology
Studentu 50, LT-3028, Lithuania
hepran@if.ktu.lt

Abstract. The paper present an approach the use Z specification language for development aggregate formal specifications. The use of Z schemas in aggregate model permits mathematically strictly define data structures used in system description. Modified aggregate approach permits to specify dynamic behaviour of distributed information systems. The presented approach permits to describe systems in different abstraction levels. It is illustrated by two specifications of commerce logistics centre which one is used for developing simulation model and other for creating of information system specification.

Keywords. Distributed systems, formal specification, Z specification language, aggregate approach, simulation, validation, logistics centre.

1 Introduction

Distributed systems arise in many applications, including telecommunications, distributed information processing, scientific computations and real-time process control.

Two properties are essential for distributed systems:

- computation activity is represented as the concurrent execution of sequential processes,
- processes communicate by passing messages.

The models of computation generally considered to be distributed are process models, in which computational activity is represented as the concurrent execution of sequential processes [1]. The process models that are most obviously distributed are ones in which process communicate by message passing: one process sends a message by adding it to a message queue, and another process receives the message, by removing it from queue. These models vary in such details as the length of the message queues and how long a delay may occur between when as message is sent and when it can be received.

Two kind of analysis are used for analysis of distributed systems: behaviour and performance analysis. All possible trajectory are analysed during behaviour analysis and it permits to check a correctness of specification. Various validation and verification methods are used for correctness analysis. During performance analysis developed specification of distributed system is executed by computer. Performance analy sis is carried out by simulation means. The main characteristics which are analysed during behaviour and performance analyses are named in Table 1.

Table 1. Analysis methods and analysed characteristics

Kind of analysis	
Behaviour	*Performance*
Used methods	
Validation and verification	*Simulation*
Analyzed characteristics	
Safety	*Lengths of queues*
Static and dynamic deadlocks	Transmission time of messages
Boundedness	Waiting time
Invariant properties	Units utilisation coefficients
…	…
Liveness	
$P \leadsto Q$	
Termination	
…	

Popular specification languages used for design of distributed systems are:

- in protocol engineering: SDL, ESTELLE, LOTOS;
- in software engineering: Z, VDM, Raise.

Mathematical methods used in specification languages are:

- extended state automata: SDL, ESTELLE;
- calculus of communicating systems: LOTOS;
- theory of sets and mathematical logis: Z, VDM;
- piece-linear aggregates (piece-linear Markov process): ESTELLE/Ag.

The suggested in the paper technique is tightly related to aggregate approach [2], which permits to integrate behaviour and performance analysis on the base of single specification. Such possibility permits to prove that developed specification is correct and to evaluate performance characteristics of an analysed system. For example, analysing telecommunication protocol it is not enough to prove correctness of its specification but also it is needed correctly to choose parameters of protocols such as timer values, buffer sizes, channels capacities, etc.

In the application of the aggregate approach for system specification, the system is represented as a set of interacting piece-linear aggregates (PLA) [2,3]. The PLA is taken as an object defined by a set of states Z, set of input signals *X*, and set of output signals *Y*. The aggregate functioning is considered in a set of time moments $t \in T$. The state $z \in Z$, the input signals $x \in X$, and the output signals $y \in Y$ are considered to be time functions. Apart from these sets, transition *H* and output *G* operators must be known as well.

The state $z \in Z$ of the piece-linear aggregate is the same as the state of a piece-linear Markov process, i.e.: $z(t) = (v(t), z_v(t))$, where $v(t)$ is a discrete state component taking values on a countable set of values; and $z_v(t)$ is a continuous component comprising of $z_{v1}(t), z_{v2}(t), \dots, z_{vk}(t)$ co-ordinates.

When there are no inputs, the state of the aggregate changes in the following manner: $v(t) = \text{const}$, $dz_v(t)/dt = -\alpha_v$, where $\alpha_v = (\alpha_{v1}, \alpha_{v2}, \ldots, \alpha_{vk})$ is a constant vector.

The state of the aggregate can change in two cases only: when an input signal arrives at the aggregate or when a continuous component acquires a definite value. The theoretical basis of piece-linear aggregates is their representation as piece-linear Markov processes.

This paper presents a technique that extends aggregate approach by means of Z specification language [4], [5], [6].

The main element of Z specification language is a schema, which is a piece of mathematical text that specifies some aspect of the analysed system. Schema contains a declaration part and a predicate part. The predicate pat expresses some requirements about the values of the variables. The predicate part of schema defines conditions that constrain the values declared in the declaration part.

The name of the schema can be used else where in specification to refer the mathematical text. The writing of specifications often involves reusing the same notion many times, and the use of names for notion many times, and the use of names for notions improves the readability of specifications, and reduces errors that arise when one specification is copied into other one context.

The use of Z schemas in aggregate model permits mathematically strictly define data structures used in system description. Main advantage of the suggested technique is its ability to specify time moments of events occurrence in a system and, at the same time, to describe complex data structures.

The structure of the paper is as follows. Section 2 is about the use of Z specifications for description of abstract data types. Aggregate specification of simulation model of commerce logistics centre presented in Sect. 3. Simulation and validation results of commerce logistics centre are given in Sect. 4. Section 5 presents the extension of simulation model of commerce logistics centre with purpose to use it for development information system specification.

2 Description of Abstract Data Types Using Z

Abstract data types, as we can see from it name, are types in which a part of it properties is not considered (either a consideration is limited only on a part of type properties).

Abstract data types have to satisfy the following requirements:

- All operations, which are possible to perform with data of that type, have to be defined in data type definition;
- User of abstract data type can operate with values of that type by using only the defined type operations. However, one cannot operate directly with their detailed representations.

Queues will be presented as abstract data description in Z. This abstract data type will be used in Sect. 2 in specification of commerce logistics centre.

Queues used in the discrete component of aggregate state are defined as abstract data type. Queue is described by schema *queue*:

$$seq\ R^+ \overset{\Delta}{=} \{ f : N \mapsto R^+ \mid dom\ f = 1..{}^{\#}f \};$$

queue

$Q : seq\ R^+$

$head : seq_1\ R^+ \rightarrow R^+$

$tail : seq_1\ R^+ \rightarrow seq\ R^+$

$size : N$

$\forall Q_i : seq\ R^+$

$head\ Q_i = Q_i(1)$

$tail\ Q_i = \lambda n : 1..({}^{\#}Q_i - 1)\ Q_i(n+1)$

$size = {}^{\#}Q$

Operations ENQ(Q_i, x) and DEQ(Q_i, y) describe processes of placement and removal elements from FIFO queue correspondingly.

$ENQ(Q_i, x)$

$queue$

$x? : R^+$

$Q_i' = Q_i \frown \langle x? \rangle$

${}^{\#}Q_i' = {}^{\#}Q_i + 1$

$DEQ(Q_i, y)$

$queue$

$y! : R^+$

$Q_i' = tail\ Q_i$

${}^{\#}Q_i' = {}^{\#}Q_i - 1$

$y! = head\ Q_i$

Operation DEQ(Q_i, k, y) describes removal of k-th element from queue.

$DEQ\ (Q_i, k, y)$

$queue$

y! : R^+

$\text{pre}(k, Q_i) = \lambda n : 1..(k-1)\ Q_i(n)$

$\text{after}(k, Q_i) = \lambda n : 1..({}^{\#}Q_i - k)\ Q_i(n+k)$

$Q_i' = \text{pre}(k, Q_i)\quad \text{after}(k, Q_i)$

y!= $Q_i(k)$

3 Aggregate Specification of a Commerce Logistics Centre

Conceptual Model. System consists of N *shopping centres, Warehouse and Producers.* The request of goods from *Shopping centre* is passed to *Warehouse* when an amount of goods in *Shopping centre* becomes less than threshold size. Each request in *Warehouse* is processed. If amount of goods in *Warehouse* is enough, then goods are delivered to *Shopping centre.* In other case request is placed to queue of requests. The functioning of *Warehouse* is the same as *Shopping centre.*

Aggregate specification consists of aggregates *Shopping_centre_*i, $i=\overline{1,N}$; *Warehouse* and *Producers* (Fig. 1).

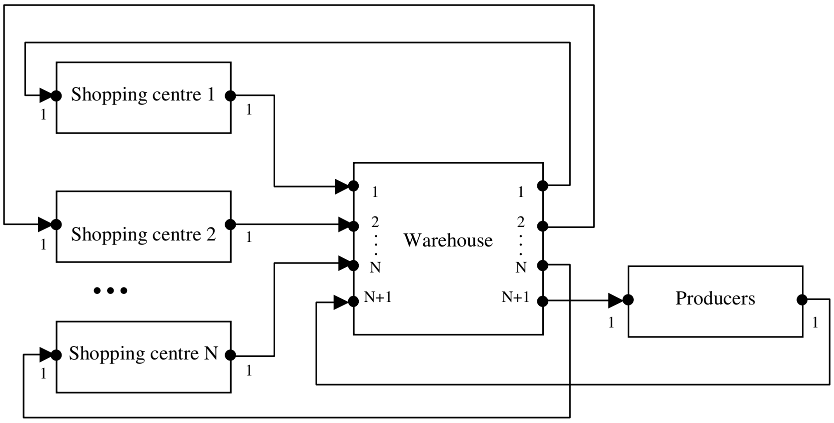


Fig. 1. Aggregate scheme of commerce logistics centre

Aggregate *Shopping centre_k*, $k=\overline{1,N}$.

1. *The set of input signals*, $X=\{x_1\}$, where $x_1=(x_{11},x_{12})$, and $x_{11}=k$, x_{12} - amount of delivered goods.
2. *The set of output signals*. $Y=\{y_1\}$, where $y_1=(y_{11},y_{12})$ and $y_{11}=k$, y_{12} - amount of requested goods.
3. *The set of external events* $E'=\{e'_1\}$, where e'_1 means that goods have been delivered.
4. *The set of internal events* $E''=\{e''_1\}$, where e''_1 means the time instant, when amount of goods at shopping centre is evaluated.
5. *Controlling sequences:*
   $e''_1 \to \{\xi_j\}$, $j=\overline{1,\infty}$, where $\{\xi_j\}$ - time interval after which amount of goods at shopping centre is evaluated.
6. *Parameters*:
   Δs - the amount of requested goods,
   s_0 - threshold. When $s(t_m)$ becomes less than s_0 request for goods is issued.
7. *Discrete component of state*:
   $v(t_m)=\{s(t_m),s_0(t_m)\}$, where $s(t_m)$ - stocks of goods in *Shopping centre*, $s_0(t_m)$ - value of threshold.
8. Continuous component of state:
   $z_v(t_m)=\{w(e''_1,t_m)\}$.
9. *Initial state*:
   $s(t_0)\geq s_0$, $w(e''_1,t_0)=t_0+\xi_1$, $s(t_0)=s_0$.

10. *Transition and output operators*:

$H(e_1'')$:

$$s(t_{m+1}) = s(t_m) + x_{12}\,.$$

$H(e_1'')$:

$$s(t_{m+1}) = \max\,(0, s(t_m) - \eta_j\,),$$

where η_j - demand for goods.

$$w(e_1'', t_{m+1}) = t_m + \xi_j\,.$$

$G(e_1')$:

$$y_1 = (k, \Delta s),\ \text{if}\ s(t_m) \geq s_0 > s(t_{m+1})\,.$$

Aggregate *Warehouse*

1. *The set of input signals* $X = \{x_1, x_2, \ldots, x_N, x_{N+1}\}$, where $x_1 = (x_{i1}, x_{i2})$, x_{i1} - number of shop, x_{i2} – amount of requested goods, x_{N+1} - amount of goods from producers.
2. *The set of output signals* $Y = \{y_1, y_2, \ldots, y_N, y_{N+1}\}$, where $y_i = (y_{i1}, y_{i2})$ and $y_{11} = k$, y_{12} - amount of requested goods.
3. *The set of external events* $E' = \{e_1', e_2', \ldots, e_N', e_{N+1}'\}$, where $e_i', i = \overline{1, N+1}$ - arrival of x_i signal.
4. *The set of internal events* $E'' = \{e_1'', e_2'', \ldots, e_N'', e_{N+1}''\}$, where $e_i'', i = \overline{1, N}$ means time instant goods delivery to *i*-th shopping centre, e_{N+1}'' means end of request processing for goods delivery.
5. *Controlling sequences:* $e_i'' \to \{\xi_{ij}\}, i = \overline{1, N}, j = \overline{1, \infty},\ e_{N+1}'' \to \{\eta_j\}, j = \overline{1, \infty}$, where ξ_{ij} - duration of time interval for goods delivery to *i*-th shopping centre; η_j - duration of time interval for request processing.
6. *Parameters*:
   Δs - the amount of requested goods, s_0 - threshold.
7. *Discrete component of state*:
   $v(t_m) = \{s(t_m), d_1(t_m), \ldots d_N(t_m), \chi(t_m), q(t_m), Q(t_m), s_0(t_m)\}$, where $s(t_m)$ - stocks of goods in *Warehouse*; $d_i(t_m)$ - amount of delivered goods to *i*-th *shopping centre*, $s_0(t_m)$ - value of threshold.

   $$\chi(t_m) = \begin{cases} 1, \text{request of goods is procesed from i - th } \textit{shopping centre}, \\ 0, \text{otherwise;} \end{cases}$$

   $q(t_m)$ - amount of goods in request.
8. *Continuous component of state*:
   $z_v(t_m) = \{w(e_1'', t_m), \ldots, w(e_N'', t_m), w(e_{N+1}'', t_m)\}$.

9. *Initial state*:

   $s(t_0) \geq s_0$, $Q(t_m) = \varnothing$, $s_0(t_0) = s_0$.

10. *Transition and output operators*:

   $H(e_1') : \; i = \overline{1, N}$.

$$s(t_{m+1}) = \begin{array}{l} s(t_m) - x_{i2}, \text{if } s(t_m) > x_{i2} \quad w(e''_{N+1}, t_m) = \infty, \\ s(t_m), \text{ otherwise;} \end{array}$$

$$Q(t_{m+1}) = \begin{array}{l} Q(t_m), \text{if } s(t_m) > x_{i2} \quad w(e''_{N+1}, t_m) = \infty, \\ ENQ(Q(t_m), (x_{i1}, x_{i2})), \text{ otherwise;} \end{array}$$

$$q(t_{m+1}) = \begin{array}{l} x_{i1}, \text{if } s(t_m) > x_{i2} \quad w(e''_{N+1}, t_m) = \infty, \\ q(t_m), \text{ otherwise;} \end{array}$$

$$w(e''_{N+1}, t_{m+1}) = \begin{array}{l} t_m + \eta, \text{if } s(t_m) > x_{i2} \quad w(e''_{N+1}, t_m) = \infty, \\ w(e''_{N+1}, t_m), \text{ otherwise;} \end{array}$$

$G(e_1')$:

$$y_{N+1} = \Delta s, \textit{if } s(t_m) \geq s_0 > s(t_{m+1}) .$$

$H(e'_{N+1})$:

$$s(t_{m+1}) = \begin{array}{l} s(t_m) + x_{N+1} - n_2, \text{if } Q(t_m) > 0 \quad w(e''_{N+1}, t_m) = \infty \quad \exists\, k > 0, \\ s(t_{m+1}) + x_{n+1}, \text{ otherwise;} \end{array}$$

$$Q(t_{m+1}) = \begin{array}{l} DEQ(Q(t_m), k), \text{if } Q(t_m) > 0 \quad w(e''_{N+1}, t_m) = \infty \quad \exists\, k > 0, \\ Q(t_m), \text{ otherwise;} \end{array}$$

$$\chi(t_{m+1}) = \begin{array}{l} n_1, \text{if } Q(t_m) > 0 \quad w(e''_{N+1}, t_m) = \infty \quad \exists\, k > 0, \\ \chi(t_m), \text{ otherwise;} \end{array}$$

$$q(t_{m+1}) = \begin{array}{l} n_2, \text{if } Q(t_m) > 0 \quad w(e''_{N+1}, t_m) = \infty \quad \exists > 0, \\ q(t_m), \text{ otherwise.} \end{array}$$

$$w(e''_{N+1}, t_{m+1}) = \begin{array}{l} t_m + n_j, \text{if } Q(t_m) > 0 \quad w(e''_{N+1}, t_m) = \infty \quad \exists > 0, \\ w(e''_{N+1}, t_m), \text{ otherwise.} \end{array}$$

$$k = \min_{1 \leq j \leq {}^{\#}Q(t_m)} \{ j \mid Q_j(t_m) = (n_1, n_2), n_2 \leq s(t_m) + \chi_{N+1} \},$$

$G(e'_{N+1})$:

$$y_{N+1} = \Delta s \text{ if } Q(t_m) > 0 \quad w(e''_{N+1}, t_m) = \infty \quad \exists k > 0 \quad s(t_m) + \chi_{N+1} - n_2 < s_0 .$$

$H(e_i'') : \; i = \overline{1, N}$.

$$d_i(t_{m+1}) = 0 .$$

$G(e_i'')$:

$$y_i = d_i(t_m) .$$

$H(e''_{N+1})$:

$$d_{\chi(t_m)}(t_{m+1}) = q(t_m),$$

$$w(e''_{\chi(t_m)}, t_{m+1}) = t_m + \xi_{\chi(t_m),j},$$

$$k = \min_{1 \le j \le {}^{\#}Q(t_m)} \{ j \mid Q_j(t_m) = (n_1, n_2), n_2 \le s(t_m) \},$$

$$(n_1, n_2) = Q_k(t_m),$$

$$\chi(t_{m+1}) = \begin{cases} n_1, & \text{if } \exists k > 0, \\ 0, & \text{otherwise;} \end{cases}$$

$$q(t_{m+1}) = \begin{cases} n_2, & \text{if } \exists k > 0, \\ 0, & \text{otherwise;} \end{cases}$$

$$w(e''_{N+1}, t_{m+1}) = \begin{cases} t_m + \eta_j, & \text{if } \exists \text{k} > 0, \\ w(e''_{N+1}, t_m), & \text{otherwise;} \end{cases}$$

$$Q(t_{m+1}) = \begin{cases} DEQ(Q(\text{t}_\text{m}), k), & \text{if } \exists \text{k} > 0, \\ Q(t_m), & \text{otherwise;} \end{cases}$$

$$s(t_{m+1}) = \begin{cases} s(t_m) - n_2, & \text{if } \exists k > 0, \\ s(t_m), & \text{otherwise.} \end{cases}$$

$G(e''_{N+1})$:

$$y_{N+1} = \Delta s, \text{ if } s(t_m) \ge s_0 > s(t_{m+1}).$$

Aggregate *Producers*

1. *The Set of Input Signals:* $X = \{x_1\}$, where x_1 - request for delivery of goods.
2. *The Set of Output Signals:* $Y = \{y_1\}$, where y_1 - amount of delivered goods.
3. *The Set of External Events*: $E' = \{e'_1\}$, where e'_1 - arrival of x_1 signal.
4. *The Set of External Events*: $E'' = \{e''_1\}$, where e''_1 - duration of time interval of goods delivering from *Producers*.
5. *Controlling Sequences*:
   $e''_1 \mapsto \{\xi_j\}$, $j = \overline{1, \infty}$
6. *Parameters*:
   $z_v(t_m) = \{w(e''_1, t_m)\}$
7. *Discrete Component of State:*
   $v(t_m) = \{d(t_m)\}$, where $d(t_m)$ - an amount goods delivered to *Warehouse*.
8. *Initial State:*
   $w(e''_1, t_0) = \infty$, $d(t_0) = 0$.

9. *Transition and Output Operators*:

$H(e'_1)$:

$w(e''_1, t_{m+1}) = t_m + \xi_j$

$d(t_{m+1}) = x_1$.

$H(e''_1)$:

$w(e''_1, t_{m+1}) = \infty$, $d(t_{m+1}) = 0$.

$G(e''_1)$:

$y_1 = d(t_m)$.

4 Results of Validation and Simulation of Commerce Logistics Centre

Formal specification presented in the previous section was used developing simulation model of commerce logistics centre. Simulation model was created using system PRANAS-2 [3], which consists of the following software tools:

- knowledge-based specification editor,
- validation subsystem,
- simulation subsystem.

Validation of specification is carried out using reachable state method that permits to analyse such system properties: boundedness, absence of specification, completeness, static and dynamic deadlocks freeness, termination.

Illustration of validation results are presented in Table 2. Numbers that are written after words *Shopping centre_i*, $i = \overline{1,3}$; *Warehouse*, *Producers* mean values of corresponding coordinates of aggregates. Table 2 illustrates alteration of discrete coordinates starting from initial state till the state when the first request for goods will be executed in warehouse.

Table 2. Illustration of validation results

```
Shopping centre_1 : 7 4
Shopping centre_2 : 7 4
Shopping centre_3 : 7 4
Warehouse : 25 0 0 0 0 0 0 16
Producers : 0
```

↓ *Event* e''_1 in *Shopping centre*_3

```
Shopping centre_1 : 7 4
Shopping centre_2 : 7 4
Shopping centre_3 : 5 4
Warehouse : 25 0 0 0 0 0 0 16
Producers : 0
```

↓ Event e''_1 in *Shopping centre*_1

```
Shopping centre_1 : 4 4
Shopping centre_2 : 7 4
Shopping centre_3 : 5 4
Warehouse : 25 0 0 0 0 0 0 16
Producers : 0
```

↓ *Event* e_1'' in *Shopping centre*_2

```
Shopping centre_1 : 4 4
Shopping centre_2 : 5 4
Shopping centre_3 : 5 4
Warehouse : 25 0 0 0 0 0 0 16
Producers : 0
```

↓ *Event* e_1'' in *Shopping centre_3*

```
Shopping centre_1 : 4 4
Shopping centre_2 : 3 4
Shopping centre_3 : 4 4
Warehouse : 25 0 0 0 2 0 0 16
Producers : 0
```

↓ *Event* e_1'' in *Shopping centre*_2

Output signal y_1 in *Shopping centre*_2

Event e_1' in *Warehouse*

```
Shopping centre_1 : 4 4
Shopping centre_2 : 3 4
Shopping centre_3 : 4 4
Warehouse : 17 0 0 0 2 8 0 16
Producers : 0
```

↓ Event e_1'' in *Warehouse*

```
Shopping centre_1 : 4 4
Shopping centre_2 : 3 4
Shopping centre_3 : 4 4
Warehouse : 17 0 8 0 0 0 0 16
Producers : 32
```

↓ *Event* e_1'' in *Shopping centre*_1

Output signal y_1 in *Shopping centre*_1

Event e_1' in *Warehous*e

Output signal y_4 in *Warehouse*

*Even*t e_1' in *Producers*

```
Shopping centre_1 : 1 4
Shopping centre_2 : 3 4
Shopping centre_3 : 4 4
Warehouse : 9 0 8 0 1 8 0 16
Producers : 32
```

The results of simulation are presented in Fig. 2. The simulation experiments have been carried out at the following parameters of the system:

Shopping centre_i, $i = \overline{1,3}$: $\mathrm{P}(\xi_{ij} = 24) = 1, \Delta s = 4, s_0 = 7$.

Warehouse: $\xi_{ij} \in [2,3], i = \overline{1,3}; \ \eta_j \in [0{,}25;0.5], \ \Delta s = 16, \ s_0 = 25$.

Producers: $\xi_j \in [24,48]$,

Results of simulation are presented in Fig. 2.

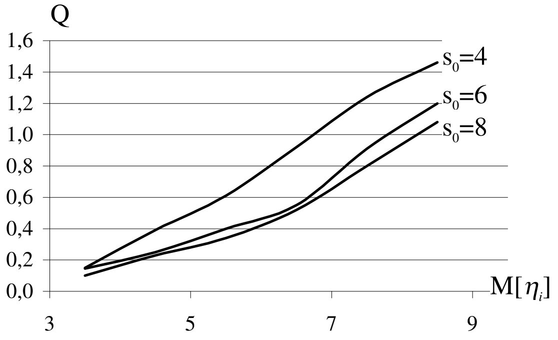


Fig. 2. dependencies of *Q* length in *Warehouse* between mean values of selling in shopping centres

5 Further Detailing of Aggregate Specification of Commerce Logistics Centre

Model presented in previous section is oriented to carry out performance analysis of commerce logistics centre. In this section it will be shown how aggregate model can be extended with the purpose to use it for development of information system. Simulation model operates with only one kind of goods. Commerce logistics centres usually operates with a lot of goods. That's why new details have to be introduced in the specification.

Extension of specification has been done for aggregate *Warehouse*. A new abstract data type *Warehouse* is introduced that has operations *get, request and delivery*.

The schema *warehouse* describes quantities of goods in warehouse (*in_stock*), the queue of requests that can not be processed due to lack of goods (*queue_requests*), threshold values for each product (*threshold_level*).

```
warehouse
in_stock : product_names ⇸ ℕ
queue_requests : seq (ℕ × product_names × ℕ)
threshold_level : product_names ⇸ ℕ
────────────────────────────
dom in_stock = dom threshold_level
```

The schema *get* describes the request operation. The input of schema is list requested goods and their quantities and output is list of goods and their quantities that will be delivered.

get

Δ warehouse
s : seq($\mathbb{N} \times$ (dom list? / dom list!) $\times \mathbb{N}$ |
#s = $^{\#}$(dom list? / dom list!)
list? : product_names $\rightarrow\!\!\!\!\!\!+\;\mathbb{N}$
list!: product_names $\rightarrow\!\!\!\!\!\!+\;\mathbb{N}$
shop? : $\mathbb{N}$

don list? $\subseteq$ in_stock
list! ={g : product_names; n : $\mathbb{N}$ |
(g $\mapsto$ n) $\in$ list? $\wedge$ in_stock(g) $\geqslant$ list?(g)}
in_stock' = in_stock $\oplus$ {g : product_names; n : $\mathbb{N}$ | g $\mapsto$ n $\in$ list! $\bullet$
g $\mapsto$ (in_stock(g) – list? (g))}
threshold_level'= threshold_level
queue_requests' = queue_requests$\frown$s

The schema *request* creates the list of requested products.

request

Ξ warehouse
list! : product_names $\rightarrow\!\!\!\!\!\!+\;\mathbb{N}$

list! : {g : product_names; n : $\mathbb{N}$ |
(in_stock g < threshold_level) $\bullet$
g $\mapsto$ (3 $\bullet$ threshold_level g)}
in_stock' = in_stock
queue_requests' = queue_requests
threshold_level' = threshold_level

The schema *delivery* creates the list of delivered products.

delivery

Δ warehouse
list? : product_names $\rightarrow\!\!\!\!\!\!+\;\mathbb{N}$

in_stock' = in_stock $\oplus$
{g : product_names; n : $\mathbb{N}$ | g $\longrightarrow$ n $\in$ list? $\bullet$
g $\mapsto$ (in_stock(g)+list? (g))}
in_stock' = in_stock
queue_requests' = queue_requests
threshold_level' = threshold_level

Correspondences between components of aggregate *Warehouse* and above introduced Z schemas are presented in Table 3.

Table 3. Correspondences between components of aggregate *Warehouse* and above introduced Z schemas

Component of aggregate Warehouse	*Z schema*
$s(t_m), Q(t_m), \mathrm{s}_0(t_m)$	Warehouse
$H(e''_{N+1}), G(e''_{N+1})$	request
$H(e'_i), G(e''_i)$	get
$G(e''_{N+1})$	delivery

6 Conclusions

Proposed. formal specification approach which use both aggregate approach and Z specification language has the following properties:

- permits on the base of single specification to carry out both performance and behaviour analysis;
- permits to specify dynamic behaviour of distributed information systems.

References

1. Lynch, N.A. Distributed algorithms. Morgan Kaufmann Publisher, San Francisco (1996)
2. Pranevicius, H. Aggregate approach for specification, validation, simulation and implementation of computer network protocols. Lectures notes in Computer Science, 502, Springer-Verlag (1991) 433-477
3. Pranevicius, H. Formal Specification of Simulation Models Using Aggregate and Z Formal Methods. Second International Conference “Simulation, Gaming, Training and Business Process Reengineering in Operations”, September 8-9, 2000, Riga, Latvia (2000) 350-354
4. Pranevicius, H., Pilkauskas, V., Chmieliauskas, A. Aggregate approach for specification and analysis of computer network protocols. Kaunas, Technologija (1994)
5. Poter, B., Sinclair, J. Till, D. An Introduction to Formal Specification and Z. Prentice Hall (1996)
6. Fenton, N., G. Hill. Systems construction and analysis: A mathematical and logical framework. McGraw Hill Book Company (1993)

Detecting Termination of Active Database Rules Using Symbolic Model Checking

Indrakshi Ray and Indrajit Ray

Computer Science Department
Colorado State University
Fort Collins, CO 80523-1873
Email: {iray, indrajit}@cs.colostate.edu

Abstract. Many algorithms have been proposed that detect non-termination of active database rules. However, most of these provide a conservative estimate – they detect all potential cases of non-termination. The onus is on the database programmers to further analyze these cases and give more definite results about non-termination. In this paper we show how the database programmer can automatically detect non-termination using an existing symbolic model checker. Our approach does not require much expertise on the part of the database programmer, and can be used to detect termination cases which the conservative approaches reject as non-terminating ones.

1 Introduction

Although active database systems [1, 7–9, 16–18] appear to be a promising technology, users are reluctant to use it because of the uncertainty associated with how a set of active database rules acting on their own will interact with each other and with other transactions [15]. To ensure that the application will behave in a predictable manner, active database applications need to be formally verified. In this paper we show how termination of active database rules can be formally verified using model checking.

Detecting termination of active database rules is, in general, an undecidable problem. Researchers have proposed sufficient conditions [2, 11, 12] that ensure termination. One such condition is the acyclicity of the triggering graph [2]. Absence of cycles in a triggering graph indicates that the rules will eventually terminate. Cycles in the triggering graph indicate potential for non-termination. In other words, if a cycle exists in the triggering graph, further analysis must be done before one can give a more definite answer about termination. In this work we show how model checking can be used to perform such analysis and give useful results about termination.

To illustrate our approach we use the SMV model checker[1] [13]. The first step involves converting the active database application to an SMV specification. The second step is to express the termination property as a CTL (Computational Tree Logic) formulae. The third step involves using the model checker to see if the property holds. The model checker performs an exhaustive search and reports whether the property holds

[1] The SMV model checker is available from `http://www.cs.cmu.edu/afs/cs/project/modck/pub/www/modck.html`

or not. If the property does not hold, the model checker provides a trace showing the violation of the property. This trace is useful for understanding and debugging purposes.

We find that our method is able to (i) predict termination of rules that are also predicted to terminate by the triggering graph approach; (ii) predict termination of rules that do terminate, but are diagnosed as possibly non-terminating by the triggering graph approach and (iii) predict non-termination for rules that are indeed non-terminating and, additionally, provide a sequence of rule execution that leads to the non-termination.

The rest of the paper is organized as follows: Section 2 discusses some related work. Section 3 gives an overview of our approach. Section 4 first presents an overview of the SMV model checker and then illustrates our approach using a motivating example. Section 5 presents a second example with terminating rule sets and the outcome of analyzing this example with SMV. Section 6 concludes this paper.

2 Related Work

Widom et al. [2] propose a static, graph theoretic approach to detect if a set of active database rules have properties, such as, termination, confluence and observable determinism. This work involves building a *triggering graph* for an active database application – the vertices correspond to the active rules of the application; edge (i, j) corresponds to the possibility that the rule represented by vertex i can trigger the rule represented by vertex j. Cycles in the graph identify potential scenarios of non-termination.

Karadimce et al. [11] argue that a simple syntactic analysis may produce a graph containing cycles whereas a more detailed analysis may produce an acyclic graph. If a detailed analysis can show that a rule p can never trigger another rule q, then the edge (p, q) can be safely removed from the graph. Removing edges in this manner may at some stage yield an acyclic graph indicating that the rules do indeed terminate.

Baralis and Widom [6] introduced another kind of graph known as *activation graph* to detect termination. The vertices corresponds to the rules of the application. An edge (i, j) signifies that the action of rule i may satisfy the condition of rule j. Acyclicity of this graph means definite termination. Baralis proposed a rule-reduction method [5] that makes use of both activation and triggering graphs. Any vertex that does not have an incoming activation or triggering edge is removed together with its outgoing edges. If all the vertices are removed in the process, the rule set is indeed terminating.

Rule analysis focussing on the termination and confluence properties have also been proposed by Van der Voort et al. [12]. The rule model used is very restrictive – rule actions can only modify data selected by the rule condition, and it appears that insertions and deletions are not allowed. Decidability results for the termination of rules has also been given by Bailey et al. [4]. In a separate work Bailey et al. [3] show how collecting semantic information can be useful in termination analysis.

3 Our Approach

Our approach is based on *model checking*. Model checkers require that the model being verified is represented as a finite state machine. Since software systems, in general, are infinite state machines, model checkers cannot be used directly to verify software. Recently researchers have shown how to build a finite state abstraction of the software system and verify it using model checking [10, 19].

We build a finite state abstraction of an active database application and then verify this abstraction using the symbolic model checker SMV. We use the input language to the SMV to specify the finite state machine. The termination properties are represented as Computational Tree Logic (CTL) (a subset of branching time temporal logic) formulae. The SMV model checker verifies the model by performing a search on the state space and comes up with the appropriate response: either it prints that the property holds or produces a counterexample illustrating the violation of the property.

We use two examples to describe our approach to rule termination analysis. The rules in the first example trigger each other indefinitely and the termination property does not hold. The result obtained in this case concurs with that obtained by applying the triggering-graph analysis of Widom et al. [2]. Next we modify the example slightly so that the rules do terminate. The corresponding triggering-graph has cycles indicating the possibility that some rules may not terminate. Our automated analysis, however, indicates that the rules do terminate.

3.1 Rule Execution Semantics

The modeling of the application in SMV is dependent on the rule execution semantics. We assume that our active database system incorporates the following rule execution semantics. Note that, our choice was arbitrary.

1. Sequential Execution: We assume a sequential mode of execution. That is, only one input transaction or active rule is processed at a time.
2. Rule Processing Granularity: The rules are processed after each occurance of a transaction. We assume that the transaction consists of a single database operation.
3. Conflict Resolution: If two or more rules are triggered at any time, the rule chosen for activation depends on the priorities specified in the rule definition. The other rules are queued up for later execution. If no priorities are specified, one rule is chosen arbitrarily for activation.
4. Iterative Rule Processing: One rule is selected and processed at a time before selecting and processing the other rule.
5. Set-Oriented Execution: We assume that a rule is executed once for all database instances triggering the rule or satisfying the rules condition.
6. Coupling Mode is Dependent-Decoupled: The rule processing takes place only after the original transaction has committed.

3.2 A Note on Notation

The notations we use are as follows. We use the **bold font** to describe database variables, sans serif font to describe the SMV variables, and *italics* to describe SMV keywords.

4 Modeling the Active Database Application

Before modeling the application, we briefly describe the SMV model checker [13].

4.1 The SMV Model Checker

The SMV program that is input to the model checker consists of four parts: (i) declarations of state variables, (ii) the initial states of the variables, (iii) the transition relations that change the state of the variables and (iv) the specification of the properties to be verified. The syntax of each part is given below.

Declaration of State Variables: The variables used in an SMV specification must be declared before use. The declaration also includes the type of the variable. A state variable can be of the following types: boolean, scalar, and fixed arrays. In the example below three variables x, y and z have been declared. The variable x is a boolean variable, that is, it can take one of two values: 0 or 1. The variable, y, is of type scalar, that is, it can take one of the three values A, B or C. The variable z can take any value in the range 1 to 5.

```
x : boolean;
y : {A, B, C};
z : 1..5;
```

State Initialization: The SMV *init* function defines the initial values of the variables. For example, suppose variable x is initialized to 0. This is specified in SMV as:

```
init(x) := 0;
```

State Transformation: The SMV *next* function defines how the next state values of the variables are computed using the current state values of the variables. Often a variable changes value depending on whether certain conditions are satisfied by the current state variables. Moreover, the value may change in different ways if different conditions are satisfied. These conditions can be specified using a *case* statement. A *case* statement returns the first expression on the right hand side of the colon (:), such that, the condition on the left hand side of the colon is true. The default case is often specified as the last expression of the *case* statement: the left hand side of the colon is a 1 and the right hand side is the default value. The example below will help illustrate the point. The next state value of the variable y is A if the current state values of x and z satisfy both the conditions x and (z = 1). The next state value of y is B if y currently equals B and z equals 2. If neither of these conditions are true, that is, in the default case, the value of y remains unchanged.

```
next(y) :=
 case
   x & (z = 1): A;
   (y = B) & ( z = 2) : B;
   1 : y;
 esac;
```

Specifications of the Properties using CTL: The properties to be verified must be specified as Computational Tree Logic (CTL) formulae. A CTL formula is a boolean expression, an existential (E) path formula, an universal (A) path formula, or the application of standard boolean operators to CTL formulae. A path formula is the application of the temporal operators next (X), eventually (F), or globally (G), to a CTL formula. For example if we want to state that it is always true that if X is true in a state, then in some future state Y will be true we specify it as follows:

```
AG(X -> EF(y))
```

4.2 Example Application with Non-Terminating Rules

The database application has a number of tables, user-defined transactions, and rules. To keep the example short, we describe only those entities that are relevant for our purpose. The two tables that are of interest are: **emp(id,rank,salary)** and **bonus(emp-id,amount)**. Table **emp** records each employee's rank and salary, table **bonus** records a bonus amount awarded to each employee.

The two transactions that we are interested in are: **updateRank** and **updateAmt**. The transaction **updateRank** increases the rank of an employee and **updateAmt** increases the bonus amount. To keep the example simple, we assume that an employee's rank is always increased by 1 and the bonus is always increased by 10.

We define two rules that trigger each other indefinitely. The first rule, **bonusRank**, states that whenever an employee's bonus is increased by 10, the employee's rank is increased by 1:

```
create rule bonusRank on bonus
when updated(amount)
   then update emp
      set rank = rank + 1
         where id in (select emp-id from new-updated, old-updated
            where new-updated.emp-id = old-updated.emp-id
                  and new-updated.amount - old-updated.amount = 10)
```

The second rule, **rankBonus**, states that whenever an employee's rank is modified, that employee's bonus is increased by 10.

```
create rule rankBonus on emp
when updated(rank)
   then update bonus
      set amount = amount + 10
         where emp-id in (select id from new-updated)
```

4.3 Converting the Database Application to an SMV Specification

A database application often has a very large number of states. To be efficiently executed by the SMV model checker the number of states must be minimized. Thus, the biggest challenge is in downsizing the application without changing its properties.

In this section we first give an algorithm for converting a database application to an SMV specification. Then, we give some hints on how to reduce the state space.

Step 1: Creation of SMV State Variables

Step 1a: For each cell in each table of the database define two SMV variables. One of these variables will contain the current state's value of the cell and the other will contain the previous state's value. As a convention the previous state variables contain the prefix 'old'. Recall that each variable in SMV must be specified with a range. For now, we assume that the range of this variable equals the domain of the values that the corresponding attribute can take. Later on, we specify how to reduce this range to minimize the state explosion problem.

State variables for the tables in the example: For each row i (where i = 1,2,3, ...) in the table **emp** we define the variables id-i, rank-i, salary-i to store the current state's values of the respective cells, and the variables oldid-i, oldrank-i, oldsalary-i to store the previous state's values. We create similar variables for the table bonus. Assuming that each table has just one row, the following variables are created. id-1 , rank-1, salary-1, oldid-1 , oldrank-1, oldsalary-1, empid-1, amount-1, oldempid-1, oldamount-1. For the sake of convenience we elimiate the suffix -1 from the variables. The declaration of the variables and their types using a pseudo SMV notation[2] is:

```
id : 0..1000;
rank : 0..no upper limit;
salary : 0..no upper limit;
oldid : 0..1000;
oldrank : 0..no upper limit;
oldsalary : 0..no upper limit;
empid  : 0..1000;
amount : 0..no upper limit;
oldempid  : 0..1000;
oldamount : 0..no upper limit;
```

Step 1b: For each rule in the application, define a SMV boolean variable. The value 1 of the variable indicates that the corresponding rule has been triggered. The value 0 indicates that the corresponding rule has not been triggered.

State variables for the rules in the example: The motivating example has two rules that generate the following state variables in SMV.

```
rankBonus : boolean;
bonusRank : boolean;
```

Step 1c: Define an SMV scalar variable active to indicate which rule is currently active. Corresponding to each rule in the application there is an enumerator. Depending on which rule is active, the variable active takes on the corresponding value. There is also another enumerator, n, that is used to denote that no rules are active in a particular state.

State variable to represent current active rule: For our example the variable active can take the value bR (indicating that the rule **bonusRank** has been chosen for activation), rB (indicating that the rule **rankBonus** has been chosen for activation), or n (indicating that no rules have been chosen for activation).

[2] In SMV we must specify a range with a lower and an upper limit

```
active : {bR, rB, n};
```

Step 1d: Define an SMV scalar variable input to indicate which input transaction is currently being processed. For each input transaction, there is a corresponding enumerator. There is further an enumerator, n, that is used to denote that no input transaction is currently being processed.

State variable to represent current active transaction: The motivating example has two transactions **updateRank** and **updateAmt**. Thus the variable input can take in any of the three values uR (indicating that the current transaction being processed is **updateRank**), uA (indicating that the current transaction being processed is **updateAmt**) and n (indicating that no input transaction is currently being processed).

```
input : { uR, uA, n};
```

Step 2: State Initialization

Step 2a: For each SMV variable derived from the database table, specify the appropriate initial values using the *init* statement. The initial values depend on the application semantics and cannot be generalized.

Initializing state variables corresponding to the table: For our motivating example, the initial values are as follows:

```
init(id) := 0;
init(rank) := 0;
init(salary) := 0;
init(oldid) := 0;
init(oldrank) := 0;
init(oldsalary) := 0;
init(empid) := 0;
init(amount):= 0;
init(oldempid) := 0;
init(oldamount) := 0;
```

Step 2b: The variables corresponding to the rules are initialized to 0. This is because initially no rules are triggered.

Initializing state variables corresponding to the rules: For our example this is done as follows:

```
init(rankBonus) := 0;
init(bonusRank) := 0;
```

Step 2c: The variables active and input are each initialized to n – indicating that none of the rules are active and no input transaction is being processed.

Initializing the variables active *and* input*:* For our example this is implemented as follows:

```
init(active) := n;
init(input) := n;
```

Step 3: State Transformation

Step 3a: The SMV variables that contain previous state's values are changed as follows: the next state values of these variables equal the current values of the corresponding variables.

State transformation of variables containing previous state values: For our example this is

```
next(oldid) := id;
next(oldrank) := rank;
next(oldsalary) := salary;
next(oldempid) := empid;
next(oldamount) := amount;
```

Step 3b: The SMV variables containing current values are modified as per application semantics. Typically, in a database application, there are transactions and/or rules that update the corresponding cells in the table. A *case* statement is used to model this. The number of expressions in the *case* statement is the number of transactions and rules that update this cell plus one for the default case.

State transformation of variables containing current state values: For our example, rank and amount are the only variables that are changed by triggers or transactions. Consider the state transformation of the variable rank given below:

```
next(rank) :=
  case
   (input = uR) : rank + 1;
   (active = bR) : rank + 1;
   1 : rank;
  esac;
```

The first expression in the *case* statement indicates that rank gets updated if the current input transaction is **updateRank**. The second expression indicates that rank gets updated if the current active rule is **bonusRank**. The last expression is the default case – if none of the above two cases are true, then rank remains unchanged.

Similarly, the variable amount's state transformation is given by the following SMV code:

```
next(amount):=
  case
    (input = uA) : amount + 10;
    (active = rB) : amount + 10;
    1 : amount;
  esac;
```

Step 3c: The rule variables are set if the corresponding rules get triggered. Note that, a rule gets triggered if some event has occured and some condition is satisfied. The event can be either an insert, update or delete operation on a table. The condition may depend on the value of some attributes. The state variables representing the rules are modified in the following way. There is *case* statement describing the different conditions in which the rule gets triggered. The first expression contains the conjunction of the events and the conditions that cause the rule to be triggered. The second expression says that if the rule is triggered, but has not been processed, then the rule remains triggered. The third expression gives the default case which means the rule variable is set to 0; this indicates that the rule will not be triggered if the above conditions are not satisfied.

State transformation of rule variables: For the rule **rankBonus** the SMV specifications is:

```
next(rankBonus) :=
  case
   !(rank = oldrank) : 1;
   (rankBonus = 1) & !(active = rB) : 1;
   1 : 0;
  esac;
```

The first expression says that the rule **rankBonus** is triggered if the value of rank changes. The second expression says that if **rankBonus** was triggered but not activated, it will be activated in the next state. Finally, the default case says that if none of the above conditions is true then the rule is not triggered.
Similarly the SMV specification for the rule **bonusRank** is:

```
next(bonusRank) :=
  case
   !(amount = oldamount) : 1;
   (bonusRank = 1) & !(active = bR) : 1;
   1 : 0;
  esac;
```

Step 3d: A *case* statement is used to specify how the value of active changes. For each rule variable there are two expressions. The first expression indicates that if a rule variable is true but the rule has not been activated, then active gets the value of that rule. The second expression says that if the event and the condition of a rule are both satisfied, the rule must be activated.
Note the order in which the expressions corresponding to the different rule variables are specified in the *case* statement, determine the priority of rule execution. This is because of the property that if the conditions for multiple expressions in a *case* statement are satisfied, only the first expression is executed.

State transformation of variable active*:* The first expression in the *case* statement says that if the current active trigger is not **bonusRank** but it is queued up for activation, then in the next state it will become active. The second expression makes a similar argument about the trigger **rankBonus**. The third expression says that if the **amount** has been changed then in the next the rule **bonusRank** will be activated. The fourth expression says that if the **rank** has been updated, then the rule **rankBonus** will be activated. If none of the above conditions are true, then no rule will be activated in the next state.

```
next(active) :=
  case
    !(active = bR) & (bonusRank = 1) : bR;
```

```
      !(active = rB) & (rankBonus = 1) : rB;
      !(amount = oldamt) : bR;
      !(rank = oldrank) : rB;
      1 : n;
    esac;
```

Step 3e: The variable input determines which transaction is being processed. We assume that once an input transaction is being processed or there is a trigger activated, no other new transaction is accepted because the processing is not yet complete. Otherwise the input can take any value from its possible enumerators. The value that input takes in this case is specified non-deterministically.

State transformation of variable input*:* For our example, the SMV specification is:

```
next(input) :=
 case
   !(input = n) : n;
   !(active = n) : n;
   (input = n) & (active = n) : {uR, uA, n};
 esac;
```

Converting Database Application to a Verifiable SMV Specification The SMV model checker checks all possible states for the satisfaction of the property. Hence, to avoid the state explosion problem it is required that the number of states be kept to a minimum level. In this section we discuss some optimizations to reduce the number of states. But before we talk about optimization, we need to elaborate on some details left out earlier.

Specify Upper Limits: In some applications no upper bound is specified for an attribute. However, in the SMV specification both the upper and the lower bound must be specified for a particular variable. To simulate the case of a variable not having an upper bound, we use the modulo operation and wrap around when the variable reaches the upper limit.

Specifying upper limits for the example: In our example application no upper limit has been specified for the rank attribute. However, the corresponding SMV variable is specified with an upper bound shown below.

```
rank : 0..4;
```

Consider the state transformation of the variable rank as discussed in Step 3b.

```
next(rank) :=
  case
   (input = uR) : rank + 1;
   (active = bR) : rank + 1;
   1 : rank;
  esac;
```

With the above example, the rank will soon reach an upper bound. To avoid this scenario, we use the modulo operation shown below:

```
next(rank) :=
  case
   (input = uR) : (rank + 1) mod 5;
   (active = bR) : (rank + 1) mod 5;
   1 : rank;
  esac;
```

Note that the modeling above does not distinguish between ranks 1, 6, 11 etc. So if the actual value of the rank is important for some application then some extra measures must be taken. For example, suppose the application requires that if the rank is 1 then an employee gets an extra bonus. To model this we need to introduce another variable rankIsOne which equals 1, if the rank is one, and 0 otherwise.

Eliminate Redundant Variables: In **Step 1** we mentioned that for each field we create two variables. Now some of the fields are never changed in an application and they are not important with respect to the property being verified. Such variables can be safely eliminated from the specification. For instance, in the motivating example, the variables id, empid, salary, are never used in the specification. So these can safely be eliminated. Some fields in a table are never updated. For these fields we do not require a variable to store the previous state values. For example, id, empid never get updated – so we can do without the variables oldid, oldempid.

Reduce the Range of Variables: Sometimes a variable can take a wide range of values. For example, the salary of an employee can take any value in the range 0 to 200000. However, specifying salary as an integer with the above range will lead to a state explosion problem.

One solution is to just list a few possible values that the salary can take as in

```
salary : {40000, 45000, 50000, 55000, 60000}
```

This solution is fine for applications in which the range is 40000 to 60000 and increments occur in 5000. However, if the application requires that an increment of 1000 be given to an employee then this cannot be modeled if we specify the salary as above.

A second solution may be to just scale down the salary:

```
salary : 0..20
```

This solution assumes that the unit is in thousands. If this solution is used, then care must be taken to divide all the usage of salary figures by 1000. This solution is used to scale down the value of amount variable in our motivating example. Note that other solutions may also be possible. The solution which must be used depends on the application.

4.4 Specifying Non-termination Properties using Computational Tree Logic

Once we have built the finite model, the next step is to specify the non-termination property using Computational Tree Logic (CTL) formulae. To show non-termination, we have to show that in the absence of input transactions, the triggers will all be reset, that is, no trigger will be active.

The CTL formula for the termination property is as follows:

```
SPEC AG((input = n)  -> AF((active = n)))
```

The above CTL formula states that it is always the case that in the absence of any input transactions, eventually none of the triggers are active. Note that all properties stated in CTL begin with the word `SPEC`. The formula `AG(f)` means that `f` holds in every state along every path. The formula `AF(f)` means that along every path there exists some future state in which `f` holds.

The output of the model checker indicate that the CTL formulae are false; that is the rules do not terminate. The model checker also provides the scenario under which the rules do not terminate. The detailed specifications is given in [14].

5 Example with Terminating Rules

To illustrate the advantage of our approach over [2], we create a second example by slightly modifying the rule **rankBonus** of the example in Section 4.2. The modified rule, which we call **rankBonus2**, is given below. As before the **rankBonus2** trigger is set when rank is updated. However, this trigger updates the amount for the employee whose **rank** is less than 3.

create rule **rankBonus2** on **emp**
when updated(**rank**)
  then update **bonus**
    set **amount** = **amount** + 10
      where **emp-id** in (select **id** from **new-updated**
            and **new-updated.rank** < 3)

The finite state model for this application is developed in the same manner as in the previous example. The model is very similar except that we introduce a boolean state variable rankLessThan3. The value of rankLessThan3 equals 1 when rank is less than 3 and 0 otherwise. Initially rankLessThan3 is 1. Whenever the current rank is greater than or equal to 2 and the current input transaction is **updateRank** or the current active trigger is **bonusRank**, rankLessThan3 is set to 0. Otherwise, the value of rankLessThan3 is not changed.

The SMV specification for the state initialization and state transformation of rankLessThan3 are given below:

```
init(rankLessThan3) := 1;
 next(rankLessThan3) :=
  case
   (input = uR) & (rank >= 2) : 0;
   (active = bR) & (rank >= 2) : 0 ;
   1 : rankLessThan3 ;
  esac;
```

The **rankBonus2** trigger is set when rank is updated. This trigger updates the amount for the employee whose rankLessThan3 is true. The SMV specification for the state initialization and state transformation of rankBonus2 is identical to that of rankBonus of the previous example. This is expected because the event and the condition part of the triggers are the same. The action parts of the two triggers are different and this is reflected in the state transformation of variable amount.

```
init(amount) := 0;
 next(amount) :=
  case
    (input = uA) : (amount + 1) mod 5;
    (active = rB) & rankLessThan3 : (amount + 1) mod 5;
    1 : amount;
  esac;
```

The CTL formulae remains the same:

```
SPEC AG((input = n)  -> AF((active = n)))
```

For this particular example, the rules **bonusRank** and **rankBonus2** are terminating. The triggering graph analysis technique [2] detects a cycle signifying a potential source of non-termination. However, when we execute the SMV program corresponding to this example, the CTL formulae return true indicating that the triggers do indeed terminate.

```
-- specification AG (input = n -> AF active = n) is true
```

6 Conclusion

Our contribution is that we show how termination of active database rules can be formally verified by the database programmer using an easy-to-use symbolic model checker. The formal analysis provides assurance that the resulting system indeed possesses the termination property. Using a very simple example, we illustrate that our approach can detect termination for a case where the triggering graph approach of Aiken et al. [2] fails.

The major difficulty is in developing an accurate finite state model of the active database application. The modeling should be such that the property being verified should not be altered in the process of developing the finite state abstraction. In this paper we illustrate manually how the model can be developed. A future work remains how some automated verification methodology can be used to create a model that can be automatically checked by the model-checker.

References

1. R. Agrawal and N. Gehani. Ode (Object Database and Environment): The Language and the Data Model. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 36–45, Portland, OR, May 1989.

2. A. Aiken, J. M. Hellerstein, and J. Widom. Static Analysis Techniques for Predicting the Behavior of Active Database Rules. *ACM Transactions on Database Systems*, 20(1):3–41, March 1995.
3. J. Bailey, L. Crnogorac, K. Ramamohanarao, and H. Søndergaard. Abstract Interpretation of Active Rules and its use in Termination Analysis. In *Proceedings of the 6th International Conference on Database Theory*, pages 188–202, Delphi, Greece, January 1997.
4. J. Bailey, G. Dong, and K. Ramamohanarao. Decidability and Undecidability Results for the Termination Problem of Active Database Rules . In *Proceedings of the 17th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pages 264–273, Seattle, Washington, June 1998.
5. E. Baralis, S. Ceri, and S. Parboschi. Improved Rule Analysis by means of Triggering and Activation Graphs. In T. Sellis, editor, *Rules in Database Systems*, volume 985 of *Lecture Notes in Computer Science*, pages 165–181. Springer-Verlag, 1995.
6. E. Baralis and J. Widom. An Algebraic Approach to Rule Analysis in Expert Database Systems. In *Proceedings of the 20th International Conference on Very Large Data Bases*, pages 475–486, Santiago, Chile, September 1994.
7. K. P. Eswaran. Specification, Implementations and Interactions of a Trigger Subsystem in an Integrated Database System. Technical Report IBM Research Report RJ 1820, IBM San Jose Research Laboratory, August 1976.
8. N. Gehani and H. V. Jagadish. Ode as an Active Database: Constraints and Triggers. In *Proceedings of the 17th International Conference on Very Large Databases*, pages 327–336, Barcelona, Spain, September 1991.
9. L. M. Haas et al. Starburst Mid-flight: As the Dust Clears. *IEEE Transactions on Knowledge and Data Engineering*, 2(1):143–160, March 1990.
10. D. Jackson. Niptick: A Checkable Specification Language. In *Proceedings of the Workshop on Formal Methods in Software Practice*, San Diego, CA, January 1996.
11. A. P. Karadimce and S. D. Urban. Refined Triggering Graphs: A Logic-Based Approach to Termination Analysis in an Active Object-Oriented Database. In *Proceedings of the 12th International Conference on Data Engineering*, pages 384–391, New Orleans, LA, February 1996.
12. L. van der Voort and A. Siebes. Termination and Confluence of Rule Execution. In *Proceedings of the 2nd International Conference on Information and Knowledge Management*, pages 245–255, Washington, DC, November 1993.
13. K. McMillan. *Symbolic Model Checking: An Approach to the State Explosion Problem*. PhD thesis, School of Computer Science, Carnegie Mellon University, 1992.
14. I. Ray and I. Ray. Detecting Termination of Active Database Rules Using Symbol Model Checking. Technical Report CIS-TR-005-2001, Computer and Information Science Dept., University of Michigan-Dearborn, May 2001. Available from `http://www.engin.umd.umich.edu/~iray/research/index.html/`.
15. E. Simon and A. Kotz-Dittrich. Promises and Realities of Active Database Systems. In *Proceedings of the 21st International Conference on Very Large Databases*, pages 642–653, Zürich, Switzerland, September 1995.
16. M. Stonebraker, L. A. Rowe, and M. Hirohama. The Implementation of POSTGRES. *IEEE Transaction on Knowledge and Data Engineering*, 2(1):125–142, March 1990.
17. J. Widom and S. Ceri. *Active Database Systems. Triggers and Rules for Advanced database Processing*. Morgan Kaufmann Publishers, Inc., San Francisco, CA, 1996.
18. J. Widom, R. Cochrane, and B. Lindsay. Implementing Set-Oriented Production Rules as an Extension to Starburst. In *Proceedings of the 17th International Conference on Very Large Databases*, Barcelona, Spain, September 1991.
19. J. M. Wing and M. Vazari-Farahani. A Case Study in Model Checking Software Systems. *Science of Computer Programming*, 28:273–299, 1997.

A Data Model for Flexible Querying

Jaroslav Pokorný[1] and Peter Vojtáš[2]

[1] Department of Software Engineering, Faculty of Mathematics and Physics
Charles University Prague
Malostranské nám. 25, 100 00 Praha 1, Czech Republic
fax (+420 2) 21914323, phone (+420 2) 21914265
`pokorny@ksi.ms.mff.cuni.cz`

[2] Department of Computer Science, Faculty of Science
P.J. Šafárik University Košice
á 5, 041 54 Košice, Slovakia
fax (+421 95) 62 221 24, phone (+421 95) 62 209 49
`vojtas@kosice.upjs.sk`

Abstract. The paper is dealing with the problem of flexible querying using vague linguistic expressions and user dependent requirements. We propose a solution based on incorporating weights into scoring rules by the usage of fuzzy logic and fuzzy similarities. We define a data model, which enables to answer queries over crisp data using fuzzy knowledge base, fuzzy interpretation of vague expressions and fuzzy similarities. We present an extension of a positive relational algebra and show that its the expressive power together with a fuzzy fixpoint operator is sufficient for evaluating fuzzy Datalog programs. We discuss also a computational model for queries with a threshold on truth values and optimization of such queries.

Keywords: flexible querying, vague linguistic expressions, fuzzy Datalog, fuzzy similarities, fuzzy relational algebra, threshold computation

1 Introduction

The information to be stored in databases in not always precise. A related issue is the handling of imperfect or flexible queries. Fuzzy sets and possibility theories are often used in this context. For databases, it usually means to apply fuzzy set-based approach in information management.

In ([BP97] the term fuzzy databases is presented in four different meanings. Here we combine a logical approach to databases, i.e. we use logical rules in many-valued-logic, with fuzzy similarity defined on attribute domains. We deal with vagueness and gradual properties of objects represented by classical (crisp) data. The aim of this paper is to build a data model to answer queries over crisp data using fuzzy knowledge base, fuzzy interpretation of vague expressions, and fuzzy similarities over respective domains. We have developed a fuzzy variant of Datalog whose procedural semantics is described via operations of a fuzzy relational algebra and a fuzzy fixpoint operator. The approach uses also a threshold computation. This feature has practical consequences for practice. We attempt to prove basic results on expressive power of this approach. Concerning a possible implementation, we use relations to store basic facts. Their truth-values can be represented, e.g., as additional columns in relational tables. The rest of a fuzzy knowledge base is represented intentionally. Hence, an implementation of this approach can be built on top of an RDBMS or deductive system with, e.g., the Datalog language. In Sect. 2 we show a motivating example for our research. In Sect. 3 we present a language for the data model. Section 4 is devoted to the definition of the fuzzy relational algebra with similarity. In Sect. 5 we briefly discuss fixpoint theory and the expressive power of our algebra. Section 6 introduces the threshold variant of the fuzzy relational algebra. The paper is closed by a discussion and conclusions in Sect. 7.

2 Motivating Example

A motivating example is inspired by a hotel reservation system of Naito et all ([Na+95]), where they described a system with logical connectives (aggregation operators) learned by a neural network for each customers preferences separately.

The illustrating example concerns a resort of High Tatras in Slovakia. We suppose the following query:

Q1. Find hotels that are near to Hágy, their cost is reasonable and their building is fine. The costs are considered to January 10th.

In order to process this query we need to specify a user's notions of being near to, cost reasonable, and building is fine.

We suppose database relations Hotel_location and Distance. Moreover, the function

$$\text{user's\_close}(x) = \max\left(0, 1 - \frac{x}{50}\right)$$

is at disposal. For example user's_close(10) = 0.8 and our system interprets the value as a truth value. For simplicity, we can imagine the representation of this function as a relation. Tuples in crisp relations have truth value 1. The notion of Near_to can be specified on the intentional level e.g. as an SQL view.

```
CREATE VIEW Near_to AS
SELECT Hotel, Business_location, TruthV
FROM Hotel_location NATURAL JOIN Distance
     NATURAL JOIN User's_close
```

The join calculates a single truth-value for the whole tuple, and Near_to is here a classical projection. Later we will see that the projection can be influenced by a confidence factor, because there can be more ways how to calculate nearness.

Hotel_location	
Hotel	Location
Szállás	Štrba

Distance		
Location	Business_location	Dist
Štrba	Hágy	10

User's_close	
Dist	TruthV
10	0.8

Hotel_location ⋈ Distance ⋈ User's_close				
Hotel	Location	Business_location	Dist	TruthV
Szállás	Štrba	Hágy	10	0.8

Near_to		
Hotel	Business_location	TruthV
Szállás	Hágy	0.8

Similar situation is for evaluating reasonability of the hotel costs. Having the user's notion for "being reasonable"

$$\text{user's\_reasonable}(x) = \begin{cases} \max(0, \frac{x}{1000}) & \text{if } x \leq 1000 \\ \max(0, 1 - \frac{x-1000}{2000}) & \text{if } x \geq 1000 \end{cases}$$

we could define the notion of cost reasonable in a similar way. The same holds for "building is fine". Now we would like to illustrate another feature of our system, namely similarity on domains. It can quite well happen that the name of the hotel in the Location database above was found in a Hungarian database but the information about the price can be in the database of a Polish operator (with a little bit different spelling).

Hotel_price		
Hotel	Time	Price
Szalas	Jan10	1600

User's_reasonable	
Price	TruthV
1600	0.7

Hotel_price ⋈ User's_reasonable			
Hotel	Time	Price	TruthV
Szalas	Jan10	1600	0.7

Cost_reasonable		
Hotel	Time	TruthV
Szalas	Jan10	0.7

Last but not least, looking for a quality of the building we use the user's notion for "being new"

$$\text{user's\_new}(x) = \max\left(0, 1 - \frac{2001 - x}{20}\right)$$

This information can be in a Slovak provider database with another spelling of the hotel name again:

Buildings	
Hotel	Year_of_constr
Salaš	1993

User's_new	
Year_of_constr	TruthV
1993	0.6

Buildings ⋈ User's_new		
Hotel	Year_of_constr	TruthV
Salaš	1993	0.6

Building_is_fine	
Hotel	TruthV
Salaš	0.6

A classical join, with the joining attribute Hotel, of the virtual relations for Near_to, Cost_reasonable and Building_is_fine would give an empty answer.

Near_to		
Hotel	Business_location	TruthV
Szállás	Hágy	0.8

Cost_reasonable		
Hotel	Time	TruthV
Szalas	Jan10	0.7

Building_is_fine	
Hotel	TruthV
Salaš	0.6

Here is the point where we enter our fuzzy similarity. Its specification can be expressed by the following table:

Similarity	Salaš	Szállás	Szalas
Salaš	1	0.6	0.7
Szállás	0.6	1	0.8
Szalas	0.7	0.8	1

Notice that dealing with the similarity in this way is not usual in classical databases. We can meet it rather in text databases or in special software, i.e. in OLAP databases or in data mining systems. Often so-called distance functions are used (e.g. Hemming distance, Levenstein distance, etc.) [BR99]. Most of fuzzy approaches solve the situation by a set-theoretic approach. They consider sets of attribute values (as subsets of respective domains), calculate their similarity degree (mainly wrt max/min operators). Consequently, the answer to the query is a set.

Our approach is based on a logical approach. We assume that our knowledge base is extended by axioms of equality and a similarity value is a product factor influencing the final answer. The product is chosen because we think that it is independent of the fact in the database whether or not there are problems in the spelling of hotel names.

Although mathematical details will be described later, we can imagine a Datalog closure of above relations wrt to fuzzy equality axioms yielding the following relations:

Near_to^1		
Hotel	Business_location	TruthV
Salaš	Hágy	.8*.6=.5
Szállás	Hágy	.8
Szalas	Hágy	.8*.8=.6

Cost_reasonable^1		
Hotel	Time	TruthV
Salaš	Jan10	.7*.7=.5
Szállás	Jan10	.7*.8=.6
Szalas	Jan10	.7

Building_is_fine^1	
Hotel	TruthV
Salaš	.6
Szállás	.6*.6=.4
Szalas	.6*.7=.4

The join of these tables in our systems is almost the classical join for all but one attribute, namely for truth value attribute.

Near_to $\bowtie_@$ Cost_reasonable $\bowtie_@$ Building_is_fine			
Hotel	Time	Business_location	Aggregated TruthVs
Salaš	Jan10	Hágy	$@(.5, .5, .6) = .5$
Szállás	Jan10	Hágy	$@(.8, .6, .4) = .7$
Szalas	Jan10	Hágy	$@(.6, .7, .4) = .6$

Query_1	
Hotel	TruthV
Salaš	.5*.8=.4
Szállás	.7*.8=.5
Szalas	.6*.8=.5

Truth value of a tuple in the joined table is calculated wrt an aggregation operator $@^\bullet$. Neither conjunction nor disjunction is appropriate to calculate this truth value. The reason is that the user's intention is that the hotel is better if it fulfils more conditions, but not necessarily all. So the conjunction would be very restrictive (and we would get possibly no answers) and disjunction would be very loose (and we would get too many answers). Here we use $@^\bullet(x, y, z) = \frac{3x+2y+z}{6}$

which gives the highest preference to distance and price and building quality are less significant (note that truth values are calculated and rounded to first decimal point).

So, to be more precise, we describe the knowledge base using Datalog/logic programming notation in a many-valued logic. The many-valued logical connectives are $\&^{\bullet}_{G}(x,y) = \min(x,y)$ and $\rightarrow^{\bullet}_{P} (x,y) = \min(1, \frac{y}{x})$ (here G stands for Goedel and P stands for Product from Goedel logic and product logic, respectively). The knowledge base is conceived as a set of rules. Our example is expressed by rules in Fig. 1. Note the first rule is considered as a root rule for the query Q1. It is equipped with a truth-value. So the answer to the query will be not a simple projection to attribute Hotel but confidence factors will be multiplied by factor 0.8. This is because there can be more rules for a user which can have different truth values.

```
query_1(Business_location, Time, Hotel)⟵P
      @(near_to(Business_location, Hotel),
         cost_reasonable(Hotel, Time),
         building_is_fine(Hotel)). with truth value = 0.8
near_to(Business_location, Hotel)⟵P
     hotel_location(Hotel, Location)&G
     distance(Location, Business_location, Distance)&G
     user's_close(Distance). with truth value = 1
cost_reasonable(Hotel, Time)⟵P
     hotel_price(Hotel, Time, Price)&G
     user's_reasonable(Price). with truth value = 1
building_is_fine(Hotel)⟵P
     buildings(Hotel, Year_of_construction)&G
     user's_new( Year_of_construction). with truth value = 1
```

Fig. 1. An example of a knowledge base

To close this introductory motivation, let us stress that our system calculates answers with truth values. This is first advantage of this many-valued approach, that the higher truth value indicates better fulfilment of the query (higher relevance of answer). Of course this is the goal of all fuzzy database and/or fuzzy query systems.

Our approach enables also threshold computations. Depending on the size of our database we can get a huge number of answers with truth value 0 (e.g. hotels more than 50km away, more expensive than 3000Sk or built more than 20 years ago. Hence we should think about to get answers as an initial segment of hotels with highest truth value. We will see later that it is difficult to describe a data model for getting the best answer or *top k answers* (because when processing the query the best answer is not necessary best along the whole way of computation, and depending on the order of evaluation it can be very low at the beginning). In this paper we concentrate on queries with thresholds. In fact, truth values have comparative character, so sometimes the best answer can have its truth value say 0.4, which does not mean it is bad, it means only that the system before finding it had to use several rules or, e.g., that there are simply no cheap hotels built in the year 2000 at distance 0.

Concerning similarity relations, these are specific for attribute domains. We considered different spelling here. There are several other sources of similarity, we do not discuss them here, see e.g. [Pe96], [Kr+98] and already mentioned distance functions. In our contribution we will assume the theory of fuzzy equality holds and all similarities are coupled with predicates by product conjunction.

Our standard reference for a database theory is [Ul89]. Flexible querying over crisp data using fuzzy techniques is considered also in [Pe96] using the minimum connective.

3 The Data Model – Language

Following [Ul89] a data model is a mathematical formalism with two parts:

1) a notation for describing data, and

2) a set of operations used to manipulate that data.

The former (both model-theoretically and proof-theoretically) will be described by fuzzy logic (see e.g. Hajek's book [Ha99], and for fuzzy logic programming [Vo98]). The latter will be presented here by new product/join, selection, union, and projection operators. Where in the two-valued case is equivalence (implications and clauses), the same variable denotes always the same value and equality means identity. In the many-valued case we have to be careful and to choose the right definition and to show it works. Nevertheless there will be a lot of analogy with the two valued case.

Language. Our language $\mathfrak{L}$ has two types of syntactical objects: logical and quantitative. The former consists of a many-sorted predicate language without function symbols (we identify here sorts for variables and attributes). The latter consists of some/all rational numbers from the unit interval $[0,1] \cap Q$. Here Q denotes the set of rational numbers.

The set of all attributes is denoted by $\mathcal{A}$. For each sort of variable (attribute) $A \in \mathcal{A}$ there is the set $\mathcal{C}^A$ of constant symbols of this sort (these are names of elements of domain of A). A predicate $r(A_1, ..., A_n)$ is defined by its pairwise different sorts of variables (attributes). For each attribute A and predicate r there is in our language at most one similarity relation $=^r_A$ interpreted as a fuzzy equivalence s^r_A on the domain $\mathcal{C}^A$.

To capture different interdependences between predicates our language can have several many valued conjunctions, disjunctions, implications and aggregations. A truth function for a conjunction & is a conjunctor $\&^\bullet : [0,1]^2 \longrightarrow [0,1]$ and for disjunction $\vee$ a disjunctor $\vee^\bullet : [0,1]^2 \longrightarrow [0,1]$ which are assumed to extend the respective two valued connectives and are monotone in both coordinates. Since no associativity, symmetry or boundary conditions are assumed, this enables us to work with approximations of connectives and/or with connectives learned from data. Truth function for an n-ary aggregation @ is an aggregation operator $@^\bullet : [0,1]^n \longrightarrow [0,1]$ which fulfils $@^\bullet(1,1,...,1) = 1$

and $@^{\bullet}(0, 0, ..., 0) = 0$. A truth function for an implication $\rightarrow$ is an implicator $\rightarrow^{\bullet}$: $[0,1]^2 \longrightarrow [0,1]$ which is non-increasing in the first (body) coordinate and non-decreasing in the second (head) coordinate and extends the two valued implication.

The logical part of our language includes classical formulas of the predicate calculus build from atoms using connectives (preserving their arity) and quantifiers.

The main syntactical objects of our language are $graded formulas$ $(\phi.\beta)$ (sometimes written (ϕ. with truth value $= \beta$)), where ϕ is a formula and β is a rational number from $[0,1] \cap Q$. Schematically

$$\text{Graded\_formulas} \subseteq \text{Formulas} \times [0,1] \cap Q$$

Structures. We base our semantics on Herbrand interpretations. A predicate (relational schema) $r(A_1, ..., A_n)$ is interpreted as a fuzzy relation $R : \mathcal{C}^{A_1} \times ... \times \mathcal{C}^{A_n} \longrightarrow [0,1]$. The fact that a tuple $(c_1, ..., c_n)$ is in R with truth value γ will be often denoted as a (crisp) tuple $(c_1, ..., c_n, \gamma)$ with an additional attribute called "truth value of R" (abbreviated $TruthV_R$).

The Herbrand base $\mathfrak{B}_{\mathfrak{L}}$ for language $\mathfrak{L}$ consists of all ground atoms (i.e. atoms without variables, only with constants). We can represent interpretations as mappings $f : \mathfrak{B}_{\mathfrak{L}} \longrightarrow [0,1]$ from Herbrand base into real numbers.

Rules. Body B is a formula built from atoms and connectives &, $\vee$, and @. Note all these connectives have monotone truth functions (we do not have negation here), hence their composition and hence also the aggregate operator are monotone.

That is why a body can be in general described as $@(B_1, ..., B_n)$, where B_i are atoms of B. This covers all types of bodies, including the classical ones.

Heads H and (database) facts F are atoms. We express logical part of rules in Datalog-like notation as $H \leftarrow B$. Note that it is an implication and not a Horn clause!

A *fuzzy Datalog rule* is a graded formula (graded implication) $(H \leftarrow B.\rho)$ and a *fuzzy Datalog fact* is a graded atom $(A.\alpha)$.

Fuzzy Knowledge Base. A *fuzzy Datalog program* is a finite set $\{(H_j \leftarrow_j B_j.\rho_j) : j \in J\} \cup \{(A_k.\alpha_k) : k \in K\}$ of fuzzy Datalog rules and facts with the additional requirement, that there are no two rules and/or facts which have syntactically identical logical parts and different quantitative part. We stress that syntactical identity is used here. There can be different rules with different truth values where the logical part are different variants of same formula (e.g. differing in constants). That is we can represent a fuzzy Datalog program as a partial syntactical mapping

$$P : Logical\_Rules \cup Atoms \longrightarrow [0,1] \cap Q.$$

The idea behind this is, that a fuzzy Datalog program is a set of requirements on truth values formulas from the logical part in models, and hence the biggest requirement is the only which counts.

An interpretation $f : \mathfrak{B}_{\mathfrak{L}} \longrightarrow [0,1]$ can be extended to all formulas calculating truth value of formulas along their complexity from truth values of atoms and truth functions of connectives. An interpretation f is a model of P if for all formulas φ we have $f(\varphi) \geq P(\varphi)$..

The theory of fuzzy logic programming for propositional logic was developed in [Vo98] and the procedural semantics was based on backward usage of modus ponens

$$\frac{(B.\beta), (H \longleftarrow_i B.\rho)}{(H.C_i(\beta, \rho))},$$

where C_i is a conjunctor residual to implicator $\longleftarrow_i$ (see [Ha99] and [Vo98]). Residual conjunctor correctly and completely evaluates this deduction rule wrt the satisfaction in structures of our fuzzy logic. Namely in an interpretation f the body B has truth value at least β and the implication $H \longleftarrow_i B$ has truth value at least ρ, then the truth value of H is at least $C_i(\beta, \rho)$ and this is the best possible promise (in the sense of extensionality of our fuzzy logic which calculates truth values of formulas along their complexity).

In the classical Datalog we work with rectified rules in order the rule evaluation is a projection of body of rules with all term equalities from the head. A rule with head $H(a_1, ..., a_h)$, where a_i's are either constants or variables, will be rewritten as $H(X_1, ..., X_h) \longleftarrow X_1 = a_1 \,\&_G\, ... \,\&_G\, X_h = a_h \,\&_G\, @(B_1, ..., B_n)$ with apropriate truth value. A generalization to more rules with the same head predicate is straightforward (see, e.g., [Ul89]).

Queries with Threshold. To keep the size of answers under control, we need thresholds for queries. The query processing with threshold t changes for different subtasks during the query evaluation as follows:

Using modus ponens, the head of the rule has the truth value $C_i(\beta, \rho) \geq t$ if the truth value of the body $\beta \geq \inf\{z : C_i(z, \rho) \geq t\} = t_0$. If this infimum exists, we continue with threshold t_0, if it does not exist we cut this branch of computation.

In the definition of fuzzy relational algebra we show how the evaluation of $@(B_1, ..., B_n)$ with threshold t_0 can be handled by assuming the threshold is a sharp threshold.

Evaluating Rules. So our task now is to describe operations of fuzzy relational algebra, such that having the above rule and having relations $R_1, ..., R_n$ currently evaluating predicates $B_1, ..., B_n$, these operations yield a relation S evaluating predicate H. A tuple $(c_1, ..., c_h, \gamma)$ is in S if there are tuples $(b^i_1, ..., b^i_{m_i}, \beta_i)$ in B_i such that when substituting tuples to respective variables then the truth value of the whole implication is at least ρ.

Similarities. We would like to emphasize, that we assume that axioms of equational theory of predicate calculus hold true and in calculation of the truth value of the rule all occurrences of equality are contributing to the final truth value. We can solve this problem by extending our fuzzy knowledge base (fuzzy Datalog program) by rules describing equational theory and properties of fuzzy equivalence. We mention the transitivity $X =^r_A Y \longleftarrow X =^r_A Z \,\&_P\, Z =^r_A Y$ and the equality axiom $r(X_1, ..., X_n) \longleftarrow X_1 =^r_1 Y_1 \,\&_P\, ... \,\&_P\, X_n =^r_n Y_n \,\&_P\, r(Y_1, ..., Y_n)$ here $=^r_1, \ldots, =^r_n$ are names for fuzzy similarities and are coupled by $\&_P$.

4 The Data Model – Fuzzy Relational Algebra

Our goal is to describe semantics of rules (without negation) by expression in a positive fuzzy relational algebra. In this section we present such an algebra as an extension of the classical positive relational algebra. Our algebra consists of selection, similarity closure, natural join, projection and union. All operations except of similarity closure are extensions of the classical relational algebra.

Selection. As already mentioned our relations have an extra attribute for truth value and expressions used in selections are allowed to use $TruthV \geq t$, e.g. $\sigma_{TruthV \geq t}(r)$. The result of such selection consists of those tuples from R which have the value of the truth value attribute at least t. This is the only extension of classical selection.

Similarity Closure. The operation which closes the relation wrt a similarity acting on a domain of an attribute is a new operation defined as follows

$$(b_1, a_2, \ldots, a_n, \beta * s^r_A(b_1, a_1)) \in \mathcal{SYM}_{s^r_A, A}(R)$$

provided A is the first attribute and $(a_1, a_2, \ldots, a_n, \beta) \in R$. Note that one step closure suffices if similarities fulfil the transitivity condition. So if an relation R is an evaluation of the predicate r, then provided equational theory of predicate calculus holds $\mathcal{SYM}_{s,A}(R)$ is an evaluation of r wrt to this theory.

Join. The operation of a natural join is in our relational algebra defined wrt to crisp equality and an aggregation operator which tells us how to calculate the truth value degree of a tuple in the join. Assume we have relations $R_1, ..., R_n$ which evaluate predicates $r_1, ..., r_n$. Moreover assume that first k-attributes in each R_i are same and $(b_1, \ldots, b_k, b^i_{k+1}, \ldots, b^i_{m_i}, \beta^i) \in R_i$ then $(b_1, \ldots, b_k, b^1_{k+1}, \ldots, b^1_{m_1}, \ldots, b^n_{k+1}, \ldots, b^n_{m_n}, @^{\bullet}(\beta^1, \ldots, \beta^n))$ is in the relation $\bowtie_{@}$ $(R_1, ..., R_n)$, that is the truth value attributes in our join do not behave as in classical join, they disappear, forwarding the respective truth values to the new aggregated truth value

Theorem 1. *The operation $\bowtie_{@}$ $(R_1, ..., R_n)$ is a sound and complete (wrt the satisfaction of fuzzy logic) evaluation of $@(r_1, ..., r_n)$ provided $R_1, ..., R_n$ were evaluations of $r_1, ..., r_n$.*

Projection. Having the evaluation R of the whole body of the rectified rule we have to use an analogy of the projection to get the evaluation of the (IDB) predicate $H(X_1, ..., X_h)$, The only difference to the two-valued case is that the truth values of tuples will change with respect to the many-valued modus ponens. So for $(b_1, ..., b_m, \beta) \in R$ the projection consists of tuples

$$(b_1, ..., b_h, C_i(\beta, \rho)) \in \Pi^{C_i(\_,\rho)}_{X_1, \ldots, X_h}(R)$$

assuming the variables selected where at the beginning of the enumeration of the tuple.

Note that similarity closure is practically projection wrt the equality rule (with the product implication and $\rho = 1$), but we consider it separately because of different procedural usage.

Union. For the case there are more rules with the same predicate in the head we have to guarantee that different witnesses to the same conclusion are not lost. In the case of different rules it means that these witnesses are aggregating together with the maximum operator (the maximal deduced value counts). Hence they unify wrt the union which calculates the maximal truth value. Assume $R_1, \ldots, R_n$ are relations with same attributes and $(b_1, \ldots, b_k, \beta_i) \in R_i$ then

$$(b_1, \ldots, b_k, \max(\beta_1, \ldots, \beta_n)) \in \bigcup_{i=1}^{n} R_i.$$

Theorem 2. *The operation*

$$\bigcup_{j=1}^{k} \Pi^{C^j(\_,\rho)}_{X_1,\ldots,X_h}(\sigma_{X_h=a_h^j}\left(\ldots\sigma_{X_1=a_1^j}\left(\bowtie_{@}(R_1^j,\ldots,R_{n_j}^j)\right)\ldots\right))$$

is a sound and complete evaluation of the $H(X_1,...,X_h)$ *(wrt the satisfaction of fuzzy logic) wrt rules ranging from* $j = 1$ *to* k

$$H(X_1,...,X_h) \longleftarrow^j X_1 = a_1^j \&_G ... \&_G X_h = a_h^j \&_G @(B_1^j,...,B_{n_j}^j).TruthV = \rho^j$$

provided $R_1^j,...,R_{n_j}^j$ *were evaluations of* $B_1^j,...,B_{n_j}^j$.

5 Fixpoint and Expressive Power of Fuzzy Relational Algebra

The problem is now to show, by analogy with the two-valued case that this evaluation procedure can be iterated to get the final relation evaluating the query with the threshold. In [Vo98] we have proved for propositional fuzzy logic programming the approximate completeness theorem based on an analogy of the fixpoint theorem, provided all connectives in body have truth functions left continuous and conjunctors evaluating modus ponens are residual to implications. We can immediately use it because we can look to ground atoms in our Herbrand base as propositional variables. Moreover the proof from [Vo98] can be generalized to the use of aggregation, because the only we need is that connectives are monotone and left continuous.

The T_P Operator. Recall the fuzzy fixpoint theory developed in [Vo98]. For fuzzy interpretations f over the Herbrand base, we can define the operator

$$T_P(f)(A) = \sup\{C_i(P(H \leftarrow_i B), f(B)) : A \text{ and } H \text{ crisp unifiable }\}$$

We showed in [Vo98] that minimal fixpoint is a model of the program P and for knowledge bases without negation the model value can be computed in an countable iteration of the T_P operator (similarly as in the crisp case) as supremum of at most countable many truth values.

This iteration of T_P will be used to prove our main theorem, namely that expressive power of fuzzy Datalog is the same as that of the relational algebra.

So if our declarative and procedural semantics is sound and complete for ground queries, it will be also for queries with free variables. It suffices for every unbound variable we can extend our language by a new constant which will be evaluated by infimum of all truth values ranging through this variable. This is again a model of our fuzzy theory and the result holds.

So our systems allows recursion and the relational algebra calculates it correctly.

Theorem 3. *Every query over the fuzzy knowledge base represented by a fuzzy Datalog program (possibly with recursion, without negation) extended by rules describing the equational theory of predicate calculus and properties of fuzzy similarities can be arbitrarily exactly evaluated by iterating described operations of fuzzy relational algebra*

Proof. Every correct answer to a query is obtained by finite iteration of the T_P operator with any prescribed precision. This operator evaluates relations in interpretation in a same way as our relational algebra does.

Note that it suffices to use similarity closure once in a row. On the other hand, it can be used at any stage of computation – in the beginning or at the very end (in our motivating example we have used it in the "middle", when the crisp join gave empty answer).

6 Threshold Computation and Query Optimization

The only substantial difference in threshold calculation is the definition of the join $\bowtie_{@}^{t_0} (R_1, ..., R_n)$ with threshold t_0 and aggregation operator $@^{\bullet}$, which is evaluating $@(B_1, ..., B_n)$ having R_i's evaluations of B_i's.

We will describe the relational operation of the fuzzy join with threshold and aggregation $\bowtie_{@}^{t_0} (R_1, ..., R_n)$ in several steps by induction through $i = 1, ..., n$.
1. calculate $t_1 = \inf\{z : @^{\bullet}(z, 1, ..., 1) \geq t_0\}$, i.e. a threshold for selection from R_1. This always exists as the infimum is through an nonempty set, because $@^{\bullet}(1, ..., 1) = 1$. Form the selection $\sigma_{TruthV>t_1}(R_1)$ and take a tuple $(b_1^1, ..., b_{m_1}^1, \beta_1) \in \sigma_{TruthV>t_1}(R_1)$.
2. Let $t_2^{\beta_1} = \inf\{z : @^{\bullet}(\beta_1, z, 1, ..., 1) \geq t_0\}$. Note that this $t_2^{\beta_1}$ always exists because $\beta_1 > t_1$ which is the infimum of such z for which $@^{\bullet}(z, 1, ..., 1) \geq t_0$ and $@^{\bullet}$ is monotone, hence at least $@^{\bullet}(\beta_1, 1, ..., 1) \geq t_0$. Form the selection $\sigma_{TruthV>t_2^{\beta_1}}(R_2)$.

Now the crucial point comes. As far as we are processing our evaluation through the aggregation and in the beginning we do not know which tuples will occur in final relation, because so far we do not know with which truth values of tuples from $R_3, ..., R_n$ will enter the construction, we have to remember both the already known truth values and their lower estimation which we get when unknown values are substituted by 1. For this we have to introduce a technical attribute called *truth value expression* and an attribute *truth value estimation* along this construction.

So for the tuple $(b_1^1, ..., b_{m_1}^1, \beta_1)$ we form a sort of product

$$P_{1to2}^{(b_1^1,...,b_{m_1}^1,\beta_1)} = (b_1^1, ..., b_{m_1}^1, \beta_1) \times \sigma_{TruthV > t_2^{\beta_1}}(R_2)$$

which is an $m_1 + m_2 + 2$-ary relation which for each tuple

$$(b_1^2, ..., b_{m_2}^2, \beta_2) \in \sigma_{TruthV > t_2^{\beta_1}}(R_2)$$

contains a tuple $(b_1^1, ..., b_{m_1}^1, b_1^2, ..., b_{m_2}^2, @(\beta_1, \beta_2, X_3, ..., X_n), @^{\bullet}(\beta_1, \beta_2, 1, ..., 1))$.
3. Now we have to join out pairs of identical attributes and the both truth value expression and truth value estimation changed by corresponding similarities. Moreover we have to select out those tuples having truth value estimation less than or equal to t_0 the original threshold for the whole body. Assume k first elements in each tuple of R_1 and R_2 correspond to the same attributes. Then we define the relation $S_{1to2}^{(b_1^1,...,b_{m_1}^1,\beta_1)}$ which for each tuple as highlighted above contains a tuple with constants $(b_1^1, ..., b_k^1, b_{k+1}^1, ..., b_{m_1}^1, b_{k+1}^2, ..., b_{m_2}^2$, and last two attributes

$$@(\beta_1, \beta_2, X_3, ..., X_n) * \prod_{i=1}^{k} s(b_i^1, b_i^2), @^{\bullet}(\beta_1, \beta_2, 1, ..., 1) * \prod_{i=1}^{k} s(b_i^1, b_i^2))$$

provided $@^{\bullet}(\beta_1, \beta_2, 1, ..., 1) * \prod_{i=1}^{k} s(b_i^1, b_i^2) \geq t_0$.

Therefore, the first step of evaluating R_1 and R_2 into $@(R_1, ..., R_n)$ provides the relation R_{1to2} which is equal to the union

$$R_{1to2} = \bigcup_{(b_1^1,...,b_{m_1}^1,\beta_1) \in \sigma_{TruthV > t_1}(R_1)} S_{1to2}^{(b_1^1,...,b_{m_1}^1,\beta_1)}$$

4. Now we proceed by induction and join R_{1to2} with R_3. For a tuple from R_{1to2} as highlighted above let

$$t_3 = \inf\{z : @^{\bullet}(\beta_1, \beta_2, 1, ..., 1) * \prod_{i=1}^{k} s(b_i^1, b_i^2) \geq t_0\}.$$

Proceed with R_{1to2} and with $\sigma_{TruthV > t_2}(R_3)$ as before and continue through all relations up to R_n.
5. Having $R_{1to\ n}$ we can omit the attribute "truth value expression", because it does not contain free variables and the relation $R_{1to\ n}$ is the evaluation of $@(B_1, ..., B_n)$ provided $R_1, ..., R_n$ were evaluations of $B_1, ..., B_n$, respectively.

Hence $\bowtie_{@}^{t_0} (R_1, ..., R_n) = R_{1to\ n}$.

Our approach offers possibilities for query optimization. First we can choose the order of relations $R_1, ..., R_n$ in which we evaluate them. We can take first the relation such that the size $\sigma_{TruthV > t_1}(R_{i_1})$ is minimal. Then having setting

$t_2 = \inf\{z : @^\bullet(t_1, 1, ..., 1) \geq t_0\}$ we work with relations in such way that the size $\sigma_{TruthV > t_2}(R_{i_2})$ is minimal, and so on.

Next in the many-valued case we can also work with incremental relations, when evaluating iteratively the intentional relations. Here $\triangle R$ are those tuples which increased their truth value strictly. All works through as in the classical case.

Our operations can be made more effective if we can build an index for the truth value attribute, it is used by every selection with $TruthV > t$.

Analogously we can introduce threshold variants of all our operations in the fuzzy relational algebra.

Theorem 4. *Every query with threshold over the fuzzy knowledge base represented by fuzzy Datalog program extended by rules describing the equational theory of predicate calculus and properties of fuzzy similarities can be evaluated by iterating threshold versions of operations of fuzzy relational algebra*

There is yet another challenge and possibility how to make this fuzzy relational algebra with similarities more efficient. We should investigate possible data structures for representing fuzzy similarities and building efficient indexing over them. To represent fuzzy similarities by a relation like here is not very effective, especially when domains are large. There are several possibilities, how to define a similarity, e.g. by a metric, by a graph, etc.

7 Discussion and Conclusion

In the literature there are several approaches to handling uncertainty and vagueness in databases and several elaborated data models.

In [Bo+99] the motivation uses also a hotel reservation example of finding a cheap and large room. Similarity is modelled by subsets of attribute domains: a fuzzy (type-2) relation R is a subset of a set product of powersets of respective domains $R \subseteq \mathcal{P}(D_1) \times \ldots \times \mathcal{P}(D_n)$. An arbitrary tuple is of the form $t = [d_1, \ldots, d_n]$ *where* $d_j \subseteq D_j$. Several formulas using min and max operators acting on similarity degrees and fuzzy membership values for data manipulation are proposed.

In [FR97] the authors model a relevance of documents in a retrieval system by probabilities and introduce a probabilistic data model. In their relational algebra they do not consider the truth value as an attribute and they do not deal with similarities. Their calculation with probabilities assigned to tuples rises from a basic probability assignment of event keys.

In [DS96] a probabilistic data model is introduced with an additional special relational operation of conditioning. Operations used to evaluate probabilities in resulting relations are based on product and probabilistic sum.

In the approach of [Ei+00] interval probabilities are assigned and a relational calculus for different probabilistic combination strategies is developed.

The work [FW00] incorporates weights into scoring rules in information retrieval using min and max operators.

In this paper we present an alternative approach. We have introduced a relational algebra based on fuzzy logic and fuzzy similarities in equational theory. In contrast to other systems we have the possibility of choosing a great variety of many-valued logical connectives to handle truth values. The Datalog model has a continuous semantics assuming all many-valued logical connectives except of implications are left continuous. This means that the Datalog fixpoint is achieved in at most countable many steps and that our fuzzy relational algebra is sufficient for evaluating the fuzzy Datalog programs.

Acknowledgements. Research supported by grants GAČR 201/00/1031 and VEGA 1/7557/20.

References

[Bo$^+$99] P. Bosc, B. B. Buckles, F. E. Petry, O. Pivert. Fuzzy databases. In Fuzzy sets in approximate reasoning and information systems. Eds. J. C. Bezdek, D. Dubois, H. Prade. Kluwer Boston, 2000, 403–468

[BP97] P. Bosc, H. Prade. An introduction to the fuzzy set and possibility theory-based treatment of soft queries and uncertain or imprecise databases, In: Uncertainty Management in Information Systems: From Needs to Solutions (A. Motro, Ph. Smets, eds.), Kluwer, 1997, 285–324.

[BR99] R. Baeza-Yates, B. Ribeiro-Neto: Modern Information Retrieval. Addison Wesley, ACM Press-New York, 1999

[DS96] D. Dey, S. Sarkar. A probabilistic relational model. ACM Trans. Database Systems 21 (1996) 339–369

[Ei$^+$00] T. Eiter, T. Lukasiewicz, M. Walter. A data model and algebra for probabilistic complex values. Infsys Research Report 1843-00-04, TU Wien, August 2000, 40 pages

[FW00] R. Fagin, E. L. Wimmers. A formula for incorporating weights into scoring rules. Theoret. Comp. Sci. 239 (2000) 309–339

[FR97] N. Fuhr, T. Roelleke. A probabilistic relational algebra for the integration of information retrieval and database systems. ACM Trans. Inf. Systems 15 (1997) 32–66

[Ha99] Hájek P. Metamathematics of fuzzy logic, Kluwer 1999

[Kr$^+$98] P. Kriško, P. Marcinčák, P. Mihók, J. Sabol, P. Vojtáš. Low retrieval remote querying dialogue with fuzzy conceptual, syntactical and linguistical unification. In Flexible Querying Answering Systems 98, T. Andreasen et al eds. LNCS 1495, Springer Berlin 1998, 215–226

[Na$^+$95] E. Naito, J. Ozawa, I. Hayashi, N. Wakami, “A proposal of a fuzzy connective with learning function”, In Fuzziness Database Management Systems, P. Bosc and J. Kaczprzyk eds. Physica Verlag, 345–364, (1995)

[Pe96] Petry F. E. Fuzzy databases - principles and applications. Kluwer 1996

[Ul89] Ullman J. D. Database and knowledge-base systems, Volumes I, Computer science Press 1988

[Vo98] P. Vojtáš. Fuzzy reasoning with tunable t-operators. J.Advanced Comp. Intelligence 2, Fuji Press (1998) 121–127

The Arc-Tree: A Novel Symmetric Access Method for Multidimensional Data

Dimitris G. Kapopoulos and Michael Hatzopoulos

Department of Informatics and Telecommunications, University of Athens
Panepistimiopolis, Ilisia 157 84, Greece
{dkapo,mike}@di.uoa.gr

Abstract. In this paper we present a novel symmetric and dynamic access method, the Arc-tree, for organizing multidimensional data. The Arc-tree uses a new space-filling curve and a partition scheme that is based on bit interleaving. It divides the data space into non-overlapping arc partitions through splits imposed by meeting planes and co-centric spheres that alternate in a fixed order. The proposed structure is k-d-cut, fixed and brickwall. The Arc-tree inco rporates the properties of metric spaces and B- trees, is independent of data distribution and excludes the storage of empty partitions. Moreover, for a given data space the partitions are identical regardless of the order of insertions and deletions. The Arc-tree arranges partitions around a starting p oint and this makes the method especially suitable for applications where distance queries and searches on planes that intersect this point are concerned. We present the Arc-tree, describe its dynamic behavior and provide algorithms that prune the set of candidates to qualify partitions for several types of queries.

1 Introduction

A wide variety of modern database applications has created a strong demand for efficiently processing of complex queries on huge databases. Thus, although there is a great range of efficient access methods, as we can see in surveys [8], [11], [21], [22], [24], the need for the development of innovative multidimensional access methods, as simple as possible, is on increase.

Many indexing schemes, such as the R-tree [12] and its variants R*-tree [2] and R^+-tree [23], use *rectangles* to restrict the area of data and organize the data space in hierarchical trees. Among others, the X-tree [3] reduces overlaps by employing super-nodes and so postpones node splitting. It also maintains the history of previous splits in order to find an appropriate split. The TV-tree [17] uses reduction of dimensionality and shifts active dimensions, ordered by importance. The A-tree tree [20] uses virtual bounding rectangles that contain and approximate minimum bounding rectangles.

Other multidimensional access methods, like the SS-tree [26], the M-tree [4] and the Slim-tree [25] employ bounding *spheres* for the shape of regions and take advantage of the characteristics of metric spaces.

Bounding spheres allows the creation of regions with short diameters, whilst bounding rectangles form regions with small volumes. In order to have the benefit of the above characteristics simultaneously, the SR-tree [15] and the G^r-tree [14] integrate bounding spheres and bounding rectangles. A region in the SR-tree consists o f the intersection of a rectangle and a sphere, whilst an active region in the G^r-tree consists of the intersection of a rectangle and an external and internal co-centric sphere.

Some indexing schemes, such as the P-tree [13] and the cell-tree [10] use *polyhedra* to shape containers. The use of additional planes results in a better approximation but increases the size of records and, consequently, decreases the fanout of the interior nodes.

Unfortunately, there is no total order that preserves spatial proximity in multidimensional spaces. In order to deal with the hard ordering problem, as efficiently as possible, *space-filling curves* impose a grid partition on the data space. Each grid cell has its own unique number that defines its position in the total order. Thus, the space-filling curve, the total order, may be organized in an one-dimensional access method. Well known space-filling curves are the row-wise enumeration of the cells [21], Gray-codes [6], the Hilbert curve [7] and the Peano curve (z-ordering) [19]. On the one hand, space-filling curves are insensitive to the number of dimensions, yet on the other, recomputing is necessary to at least one index when joining incompatible partitions of two indices.

The G-tree [16] is based on the z-ordering and the BD-tree [5]. It uses a variable length partition numbering scheme and combines the features of both B-trees and grid files [18]. Another data structure similar to the BD-tree is the BANG file [9]. Grid files divide a data space into a grid structure by splitting each dimension into several non-uniformly spaced intervals. Hence, potential values of key sets are regarded as points in a multidimensional Cartesian product space. •n access method that employs the features of z-ordering and B-trees in a manner that aims at preserving clustering, is the UB-tree [1].

In this work, we introduce the *Arc-tree*, a symmetric and dynamic multidimensional access method that is based on a new space-filling curve and aims at efficient manipulation of large amounts of point data. The rest of the paper is organized as follows: Section 2 introduces the Arc curve and presents the partition scheme and the structure of the Arc-tree. Section 3 deals with the procedures of insertion and deletion in the Arc-tree and Sect. 4 explains the searching algorithms for several types of queries. Section 5 concludes this work with a summary and future research directions.

2 The Arc-T ree

The proposed structure uses the properties of metric spaces. In order to organize a data space S through the Arc-tree, S must be considered as a *metric space* $M = (S,d)$, where d is a *distance* or *metric function*. A distance function has the properties of symmetry, non-negativity and triangle inequality.

The k-dimensional space S^k of the Arc-tree is a subset of the Euclidean space R^k, $k \geq 1$. We associate the *norm* of the difference of two points as the metric function for this space i.e.,

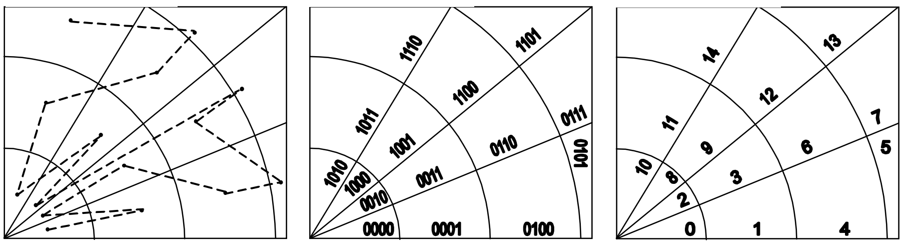
$$d(x,y) = |x-y| = \left(\sum_{j=1}^{k}(x_j - y_j)^2\right)^{1/2}, \quad \forall x,y \in R^k \tag{1}$$

We assume that the data in each dimension are bounded i.e., for each point $x = (x_1,x_2,\ldots,x_k)$ we have that $l_{c_j} \le x_j \le h_{c_j}$, $1 \le j \le k$, where $l_c = (l_{c_1},l_{c_2},\ldots,l_{c_k})$ and $h_c = (h_{c_1},h_{c_2},\ldots,h_{c_k})$ are the leftmost and rightmost points of S^k.

The Arc-tree uses a new spac e-filling curve, the *Arc curve*, to partition a multidimensional space. This curve, unlike others, does not use a grid to partition a data space. A data space is divided by the Arc curve into non-overlapping arc partitions of variable size.

Each one of the partitions is represented by a unique bit string, which resembles the identification of Peano regions. Those bit strings are totally ordered and thus can be stored in a B$^+$-tree. There is a one to one correspondence between partitions and data blocks. Each partition contains no more than a maximum number of *BC* (block capacity) entries. A full partition is split into two sub-partitions. The Arc-tree e x-cludes the possibility of storing empty partitions and is designed to adapt well to dynamic environments.

Figure 1a represents the Arc curve for some partitions in a two-dimensional space. Figure 1b shows the bit strings for these partitions and Fig. 1c, the equivalent binary representation of the partition numbers in decimal form.



(a) (b) (c)

Fig. 1. Arc curve and partition numbers in a two-dimensional space

The partition scheme of the Arc-tree divides the data space into non -overlapping arc partitions through splits imposed by meeting planes and co-centric spheres. The structure employs bit interleaving. Throughout this paper we use the words plane and sphere for hyper-plane and hyper-sphere, respectively.

A k-dimensional space is split recursively by (k-1)-dimensional planes and k-dimensional spheres. We use $\sum_{j=1}^{k-1} j$ directions of planes and one direction of sphere that alternate in a fixed order among the

$$alt = 1 + \sum_{j=1}^{k\text{-}1} j \tag{2}$$

possibilities. A splitting plan passes from a fixed line that belongs to the plane of two dimensions and includes the starting point. Initially, this line is the diagonal of the rectangle that is formed by the two dimensions. A splitting plan halves a partition into two equal sub-partitions. Before we define a splitting sphere, we proceed with some definitions and notations.

A *closed sphere* $\overline{S}(cp, r)$ and an *open sphere* $S(cp,r)$ with central point cp and radius r in S^k, are defined as follows

$$\overline{S}(cp, r) = \left\{ x \in S^k : d(x, cp) \leq r \right\} \tag{3}$$

$$S(cp, r) = \left\{ x \in S^k : d(x, cp) < r \right\} \tag{4}$$

We use the notation r_s for the radius that resides on the left of r and it is closest to it. That is, there does not exist radius r' in the Arc-tree such that $r_s < r' < r$. If r is the smallest radius, then we define $r_s = 0$. The similar notation r_b is used for the radius that resides on the right of r and is closest to it. Thus, there does not exist radius r' in the Arc-tree where $r < r' < r_b$. If r is the maximum radius, then we define $r_b = d(l_c,h_c)$. We choose the point l_c to be the central point of the spheres. That is, $cp = l_c$.

The radius of a splitting sphere $S(cp,r)$ is

$$r = \frac{r_s + r_b}{2} \tag{5}$$

Initially, it is $r_s = 0$ and $r_b = d(l_c,h_c)$.

A *partition P* imposed by the Arc-tree is a subset of S^k and is defined as the intersection of two segments.

$$P = R(P) \cap H(P) \tag{6}$$

The segment $R(P)$ is the complement of an open sphere $S(cp,r_s)$ to a co-centric closed sphere $\overline{S}(cp, r)$. That is

$$R(P) = \overline{S}(cp, r) - S(cp, r_s) \tag{7}$$

The segment $H(P)$ is a space that is included by two planes.

A split of a full partition is triggered by the insertion of a new point. The parent partition P is replaced by its offspring partitions. It follows the redistribution of data points. Data blocks are written on disk and the index is updated with the new partition numbers. There is no need to store splitting points, as they are computed from the partition numbers. Only non-empty partitions, which span the data space, are stored in the index. From the above splitting policy, we conclude that the Arc-tree is k -d-cut, fixed and brickwall.

Successive parts of Fig. 2 are examples of how partitions are split in the case of $k = 2$. For a better representation, we can assume $BC = 2$. Initially, as shown in Fig. 2a, the data space is divided into two equal size partitions by splitting along the diagonal that connects the point l_c and h_c. The first sub-partitions of the data space are numbered 0 and 1. The insertion of a third point into partition 1, results in a split. In Fig. 2b, a sector of the sphere

$$S\left(l_c, \frac{r_s + r_b}{2}\right) = S\left(l_c, \frac{d(l_c, h_c)}{2}\right) \tag{8}$$

splits the parent partition 1 and creates the children partitions 10 and 11. Each child is the complement of its brother. The *comp*(*P*) (complement) of a partition *P* is settled by inverting the least significant bit of *P*, whereas the *parent*(*P*) is settled by removing the least significant bit from *P*. When the partition 11 becomes full, a part of the diagonal that connects the point

$$l_c \text{ and } \left(\frac{l_{c_1}+h_{c_1}}{2}, h_{c_2}\right) \tag{9}$$

halves this partition into sub-partitions 110 and 111 (Fig. 2c). When the partition 110 becomes full, the sector of the sphere

$$S\left(l_c, \frac{3 * d(l_c, h_c)}{4}\right) \tag{10}$$

that belongs to the partition 110 creates the sub-partitions 1100 and 1101 (Fig. 2d). Figures 2e and 2f show two additional splits. The dash line in Fig. 2f depicts the Arc curve.

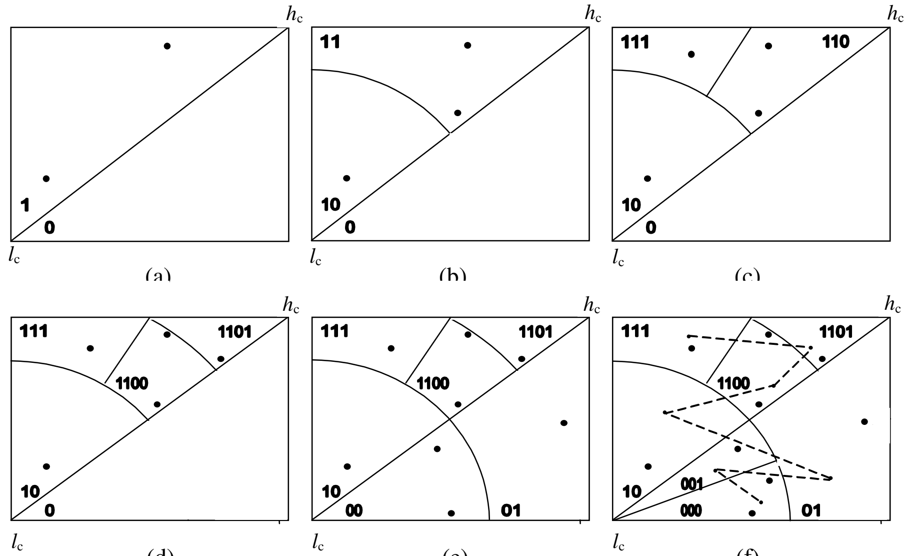


Fig. 2. Splits in a two-dimensional space

Figure 3 shows some splits in a three-dimensional space. The first three parts of the figure, show splits imposed by the planes that are perpendicular to the diagonals of the dimension pairs (d_1,d_2), (d_1,d_3) and (d_2,d_3), respectively. These planes divide the regions they cross into equal sub-regions. The notation d_i, $1 \le i \le k$ stands for the *i*-dimension. The latter part of this figure shows a split imposed by a sphere. For the clarity of presentation we do not show points inside partitions. Continuous lines for the dimensions and dash lines for the planes and the sphere are used.

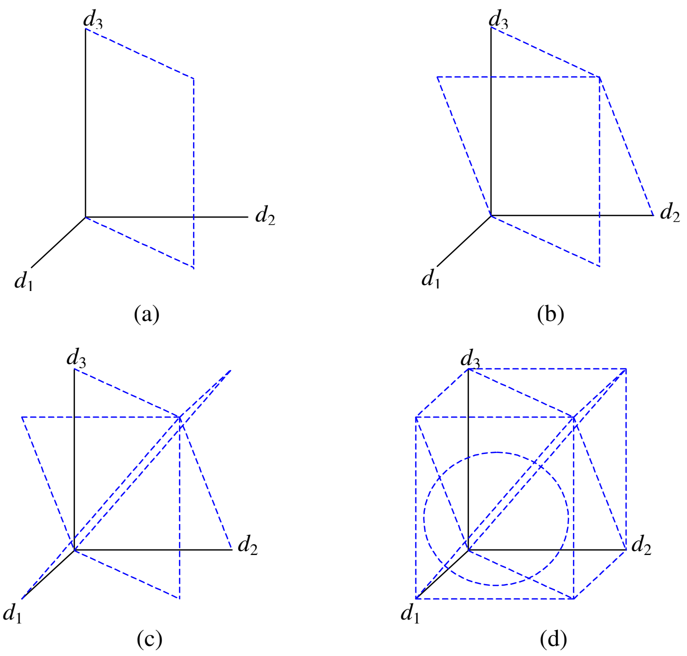


Fig. 3. Splits in a three-dimensional space

The partition scheme of the Arc-tree aims at placing as accurately as possible, pa r-titions that are spatially successive in a multidimensional space in successive places in the index. Partition numbers are stored in a B$^+$-tree. We use links from right to left in the leaf level of the index in order to facilitate sequential processing when searching for all the answers, after the first match. The Arc-tree demands a small storage space because an entry in its block is just a pair of a partition number (bit string) and an address (pointer). Addresses in the leaf level and internal nodes refer to data blocks and nodes of some lower level, respectively.

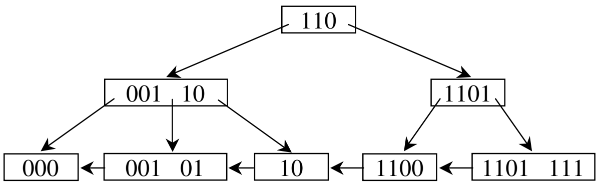


Fig. 4. The Arc-tree for the part itions of Fig. 2f

Figure 4 shows the Arc-tree that co rresponds to the partitions of Fig. 2f. The active partition numbers, which form the partition scheme, are sorted in lexicographic order at the leaf level of the tree. This means that 1101 < 111. Entries in internal levels are merely separators. They may exist from previous splits and thus may not be active partition numbers.

The Arc-tree preserves symmetry as it arranges partitions around a starting point. This makes the method particularly suitable for applications where distance queries and searches on planes that intersect this point are concerned. Until now, we have used as this point, the leftmost point of the data space. This is not a restrictive constraint. A starting point may be any point of special interest. For example, in geographic and astronomy applications where the earth is considered the center of the universe, and distance queries as well as queries based on longitude, latitude and altitude are of special interest, the starting point could be the center of the data space.

Figures 5a and 5b show partitions and the Arc curve, respectively in a two-dimensional space with uniform distribution. The starting point of the organization is the center of the data space.

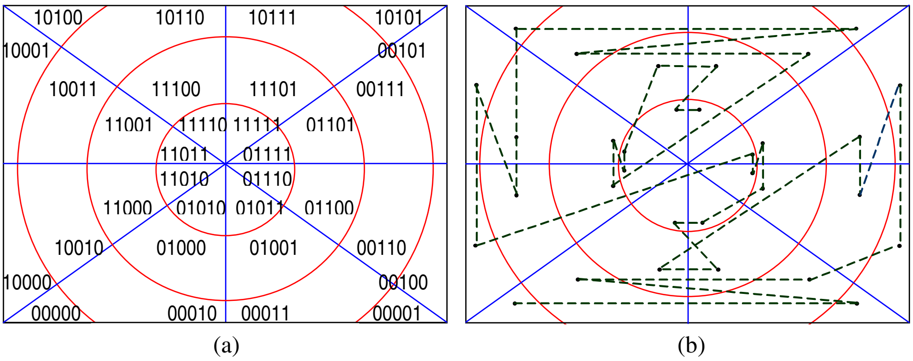


Fig. 5. Partitions and their Arc curve in a two-dimensional space with uniform distribution

3 Update Operations

The Arc-tree adapts its shape to large dynamic volumes of data, no matter what their distribution and does not need reorganization. Its partitioning scheme for a given data space is identical regardless of the order of insertions and deletions. Following, we will describe the procedures for inserting and deleting multidimensional points from the Arc-tree.

3.1 Insertion

To insert a point p in the file, we first compute its initial partition P_{init} and then the actual partition P_{act} where p should be inserted. The length of the bit string of P_{init} is equal to b, where b is the maximum number of bits of the partitions that exist in the Arc- tree. We traverse from the top, down the Arc-tree to reach the leaf level searc h-ing for P_{init}. If it exists, we set $P_{\text{act}} = P_{\text{init}}$. Otherwise, we search for the first ancestor of P_{init} moving from right to left at leaf level.

If such an ancestor exists, then P_{act} is set to be equal to it. The data block of P_{act} is transferred into main memory and if there is space, we insert p and write the block back on disk. If the block is full, we have a case of overflow and we split the block, as described in the previous section. There is no need to store any meta-information regarding splitting.

If an ancestor of P_{init} does not exist, we will find either a partition that is not a super-partition of P_{init} or we will reach the first partition of the index. Then, P_{act} is set to be the largest ancestor of P_{init} that does not overlap with an existing partition in the index. Afterwards, we insert P_{act} into the Arc-tree and place the point p into a new data block that is aligned to P_{act}.

If the attributes of two points are integers or real numbers, then the calculation of their distance is obvious. For strings we use an algorithm that transforms them to unsigned long integers. Thus, distance calculations are arithmetic for all data types. There is a slight possibility of two different strings to be transformed into the same long integer. The likelihood of an impossible partition split due to the mapping of all data points to the same point is remote. The greater the number of dimensions, the smaller the possibility. The case of an impossible split is handled by overflow data blocks.

3.2 Deletion

For the deletion of a point p, we first compute P_{init} and then check if P_{act} exists in the index. If so, we delete the point and rewrite the data block on disk. In order to have blocks filled over a threshold, we try to merge a partition P with *comp*(P) if after a deletion from P it is left with less than half points. Thus, if the sum of the number of points of P and *comp*(P) does not exceed the block capacity, we merge the underflow data block of P with the block of *comp*(P) by moving all data to one block. Consequently, *parent*(P) replaces P and *comp*(P) in the index. If *comp*(P) does not exist in the index, *parent*(P) replaces P, and the merging procedure continues in order to obtain larger and more full partitions.

4 Search

In this section we examine how several types of queries are processed using the Arc-tree.

The Arc-tree efficiently handles distance queries of the type '*find all points p*: $dist_l \le d(p,l_c) \le dist_h$', where the distances $dist_l$ and $dist_h$ are real numbers. Other expressions based on the operands =, <, ≤, >, and ≥, are sub-cases of the above general query form. The Arc-tree takes advantage of the splits imposed by spheres and r e-stricts the number of partitions to be examined. First, we evaluate two mask bit strings *msl* and *msh* through $dist_l$ and $dist_h$, respectively. As far as splits imposed by spheres are concerned, these strings declare the lower and higher position that the searched points could have in the data space, respectively. The above strings are composed of $\lfloor b/alt \rfloor$ bits. This means that the number of bits in the above mask strings is equal to the number of splits that have been imposed by spheres in a partition with length equal to b. That is, the number of this partition has maximum length among existing partitions. The algorithm in pseudo code for the creation of these mask strings is listed below.

```
ms_spheres(dist,b)
{
  r=d(lc,hc)/2;                   // sphere radius
  ms="";                          // null string
  for (i=1;i<=⌊b/alt⌋;i++)
    if (dist>r) {
      concatenate "1" to ms;
      r=3*r/2;
    }
    else {
      concatenate "0" to ms;
      r=r/2;
    }
  return ms;
}
```

The Arc-tree is searched in the range $[L_{\text{act}},H_{\text{act}}]$, where L_{act} and H_{act} are the actual partitions of the points that belong to the initial partitions L_{init} and H_{init} with numbers $l_1l_2\ldots l_b$ and $h_1h_2\ldots h_b$, respectively. The creation of L_{init} is based on *msl*, whereas H_{init} is based on *msh*. L_{act} has the smallest partition number that may qualify the query, whereas H_{act} has the biggest. It is

$$l_j = \begin{cases} 0, & \text{if } (j \bmod alt) \neq 0 \\ msl_{j/alt}, & \text{if } (j \bmod alt) = 0 \end{cases}, \ 1 \leq j \leq b \tag{11}$$

$$h_j = \begin{cases} 1, & \text{if } (j \bmod alt) \neq 0 \\ msh_{j/alt}, & \text{if } (j \bmod alt) = 0 \end{cases}, \ 1 \leq j \leq b \tag{12}$$

A partition $P \in [L_{\text{act}},H_{\text{act}}]$ is a candidate to qualify the query. Let $b(P)$ be the number of bits of P. If p_j, $1 \leq j$, is the j^{th} bit of the partition number of P, then we create a new partition P' whose number of bits is not greater than the number of bits of *msl* and *msh*. Its j^{th} bit p'_j is

$$p'_j = p_{j*alt}, \quad 1 \leq j \leq \left\lfloor \frac{b(P)}{alt} \right\rfloor \tag{13}$$

Only the data blocks of the partitions P with P' such that

$$msl \leq P' \leq msh \qquad msl \quad P' \tag{14}$$

are examined for qualified points. The operation   stands for sub-partition. We refer to the above qualification as MSQ (mask string qualification). If it is $b(P) < alt$ then MSQ is not an issue.

Figure 6 is an example of the distance query 'find all points p: $11 \leq d(p,l_c) \leq 18$' applied to the partitions of Fig. 2f. We suppose that $d(l_c,h_c) = 20$. The query region is the shadow roll that is included between the dash arcs. It is $b = 4$, $msl = 10$, $msh = 11$, $L_{\text{init}} = 0100$, $H_{\text{init}} = 1111$, $L_{\text{act}} = 01$ and $H_{\text{act}} = 111$. The partition numbers that belong to $[L_{\text{act}},H_{\text{act}}]$ are 01, 10, 1100, 1101 and 111. Although the partition 10 is crossed by the

Arc curve that connects the partitions 01 and 111, it is not examined any more as it does not satisfy the MSQ. The data blocks of 01, 1100, 1101 and 111 are examined for qualified points.

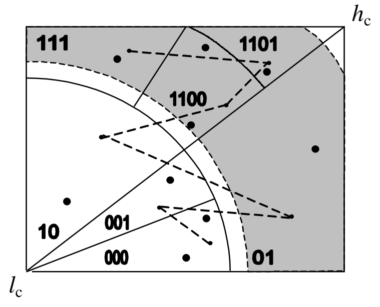


Fig. 6. A distance query for the partitions of Fig. 2f

We will now examine the query '*find all points p, which belong in a level included between two lines that cross l_c*'. Sub-cases of the above query are 'find the points, which are between and not on the lines' and 'find the points on a line'.

A line in a k-dimensional space is described by the set of the following (k-1) equations.

$$d_{j+1} = a_{j-1} * d_j, \quad a_{j-1} \in R,\ 1 \le j \le k-1 \tag{15}$$

The real number a_{j-1}, $1 \le j \le k$-1 expresses the tangent of the corner that forms the dimension d_j, with the projection of the line at the level of the dimensions d_j and d_{j+1}.

In order to answer the above query efficiently, the Arc-tree takes advantage of the splits that have been imposed by the planes. First, we evaluate the mask bit strings *msl* and *msh* which, in the current type of query, declare the lower and higher position of possible qualified points, as far as splits imposed by planes are concerned. These strings are composed of (b - $\lfloor b/alt \rfloor$) bits. That is, the length of their partition number is equal to the number of splits imposed by planes in a partition with its length equal to b. *msl* and *msh* are created through the set of equations of the two lines that stand for the lower and higher limit of the query, respectively. The algorithm is listed below.

```
ms_planes(a[k-1],b)
{
  n=0;
  for (i=1;i<k;i++)
    for (j=i+1;j<=k;j++)
      t[n++]=($h_{c_j}$-$l_{c_j}$)/($h_{c_i}$-$l_{c_i}$);       // tangent by first
         // splitting plane projection in level of $d_i$,$d_j$
  ms="";                                          // null string
  m=0;
  while (m<b-$\lfloor$b/alt$\rfloor$) {
    n=fmod(m++,alt-1);
    if (n==0) {      // new cycle of splits is examined
      i=1;                                     // dimensions of
      j=i+1; }                          // splitting plane
    else
      if (j==k)
        j=++i+1;
      else
        j++;
    f= $_{q=i}^{j-1}$ a[q-1];  // coefficient of the line for $d_i$,$d_j$
    if (f>t[n]) {          // find the sector of the line
      concatenate "1" to ms;
      t[n]=3*t[n]/2; }
    else {
      concatenate "0" to ms;
      t[n]=t[n]/2; }
  }
  return ms;
}
```

For this type of query we have

$$l_j = \begin{matrix} 0, & \text{if} \quad (j \bmod alt) = 0 \\ msl_{j-(j \operatorname{div} alt)}, & \text{if} \quad (j \bmod alt) \neq 0 \end{matrix} \quad , 1 \leq j \leq b \tag{16}$$

$$h_j = \begin{matrix} 1, & \text{if} \quad (j \bmod alt) = 0 \\ msh_{j-(j \operatorname{div} alt)}, & \text{if} \quad (j \bmod alt) \neq 0 \end{matrix} \quad , 1 \leq j \leq b \tag{17}$$

The function `div` stands for quotient. We define the bits of the partition $P^{'}$ as

$$p^{'}_{j-(j \operatorname{div} alt)} = p_j, \quad 1 \leq j \leq b \qquad (j \bmod alt) \neq 0 \tag{18}$$

As happens in the previous type of query, only the data blocks of the partitions P with $P^{'}$ satisfying the condition MSQ are examined for qualified points.

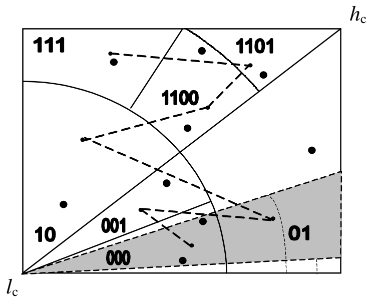


Fig. 7. A query region included between lines for the partitions of Fig. 2f

Figure 7 is an example of the query 'find all points p, which are between the lines $d_2 = \tan(4^o)*d_1$ and $d_2 = \tan(18^o)*d_1$' applied to the partitions of Fig. 2f. The query region is restricted by the dash lines and forms the shadow triangle. We suppose that the three partition lines in this figure have coefficients $\tan(22^o)$, $\tan(40^o)$ and $\tan(58^o)$, respectively. As it is, tan(4°) = 0.07, tan(18°) = 0.32, tan(22°) = 0.4, tan(40°) = 0.84, tan(58°) = 1.6 and $b = 4$, we have msl = 00, msh = 00, L_{init} = 0000, H_{init} = 0101, L_{act} = 000 and H_{act} = 01. The partition numbers that belong to $[L_{\text{act}},H_{\text{act}}]$ are 000, 001 and 01. Only the data blocks of the partitions 000 and 01 that satisfy the condition MSQ are examined for qualified points.

A *range query* is equivalent to a rectangle RQ in S^k. The number of corners of RQ is at most 2^k. In order to answer a range query, we first estimate the initial partitions of the corners of RQ. Let L_{init} and H_{init} be the lower and higher partition among these initial partitions. Afterwards, we estimate the actual partitions L_{act} and H_{act} that represent the lower and higher existing partition among all the corners. Partitions in the range $[L_{\text{act}},H_{\text{act}}]$ are candidates to qualify. The previously shown algorithms that take advantage of splits imposed by spheres and planes, prune candidate partitions.

First, we create the mask strings *msl* and *msh* through the call of `ms_spheres(d(lc,lc(RQ)),b)` and `ms_spheres(d(lc,hc(RQ)),b)`, respectively. $l_c(RQ)$ and $h_c(RQ)$ are the leftmost and rightmost corners of RQ. For each partition P in $[L_{\text{act}},H_{\text{act}}]$ we create the partition P', as shown in distance queries, and we check if it satisfies the condition MSQ. If not, the partition is excluded from further consideration.

Second, for each corner of RQ we estimate the line that connects it with the starting point l_c. Afterwards, we use the function `ms_planes(a[k-1],b)` to compute the mask strings of all corners. We are reminded that a_{j-1}, $1 \le j \le k$-1 is the tangent of the corner that forms the dimension d_j with the projection of the line at the level of the dimensions d_j and d_{j+1}. We set *msl* and *msh* to be the minimum and the maximum mask bit string, respectively. Candidate partitions are pruned in a second filter step with the condition MSQ. Only regarding the remaining partitions do we access their data block to examine for qualified data.

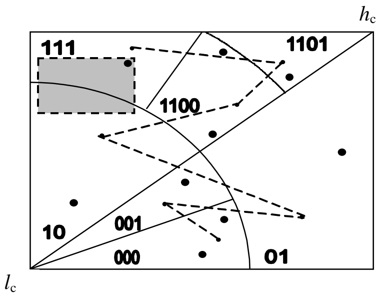


Fig. 8. A range query for the partitions of Fig. 2f

Figure 8 is an example of a range query RQ, shown by the shadow rectangle formed by dash lines, for the partitions of our running example. The initial partitions resulting from the examination of the corners of RQ are L_{init} = 1011 and H_{init} = 1110. Thus, we have L_{act} = 10 and H_{act} = 111. As a result, the partitions in the range [10,111] are candidate to qualify. These partitions are 10, 1100, 1101 and 111.

The use of `ms_spheres` for the leftmost and rightmost point of RQ gives msl = 01 and msh = 10, respectively. Thus, MSQ excludes 1101 from further consideration. The use of `ms_planes` for the lines that connect the corners of RQ with l_c results to msl = 11 and msh = 11. This means that MSQ excludes 1100 from the set of candidate partitions. Therefore, only the data blocks of the partitions 10 and 111 are examined for qualified data.

The procedures to answer a *partial match* and an *exact match query* are similar to that of a range query, as the sets of partial match and exact match queries are subsets of range queries. If j, $1 \leq j \leq k$, is a dimension that participates in a partial match query with free value, then the upper and lower corners of RQ, as regards this dimension, possess the upper and lower value of d_i, respectively. In an exact match query, RQ degenerates to a point p. We first compute P_{init} and then examine if P_{act} exists in the index. If so, we access the data block of P_{act} and examine if p belongs there.

5 Summary

In this paper, we presented the Arc-tree, a novel balanced and dynamic access method for the manipulation of multidimensional data. The proposed structure is based on the Arc curve. This is a new space-filling curve that, unlike others, does not use a grid to partition a data space.

The Arc-tree divides the data space into non-overlapping arc partitions through splits imposed by meeting planes and co-centric spheres that alternate in a fixed order. It employs bit interleaving and combines the features of metric spaces and B-trees. It excludes the storage of empty partitions and is independent of data distribution. Moreover, for a given data space, the partitions are identical regardless of the order of insertions and deletions. As far as its splitting policy is concerned, it is k-d-cut, fixed and brickwall.

The Arc-tree preserves symmetry as it arranges partitions around a starting point. This makes the method especially suitable for applications where distance queries and searches on planes that intersect this point are concerned. The set of possible qualified partitions is restricted with the use of mask strings that take advantage of the manner by which splits have been imposed by planes and spheres.

We presented algorithms for inserting, deleting and searching data from the Arc-tree. We believe that the Arc-tree is a promising access method capable at achieving its design motivation, which is to handle large quantities of multidimensional data efficiently.

Work in the pipeline includes the confirmation of the efficiency of the Arc-tree via experimental results and its extension in order to handle complex spatial objects.

References

1. Bayer, R.: The Universal B-tree for Multidimensional Indexing: General Concepts. In: World-Wide Computing and Its Applications. Lecture Notes in Computer Science, Springer-Verlag, Tsukuba Japan (1997) 10-11
2. Beckmann, N., Kriegel, H-P., Schneider, R., Seeger, B.: The R*-tree: An Efficient and Robust Access Method for Points and Rectangles. In Proc. ACM SIGMOD, (1990) 322-331
3. Berchtold, S., Keim, D.A., Kriegel, H-P.: The X-tree: An Index Structure for High-Dimensional Data. In Proc. 22nd Int. Conf. on VLDB, (1996) 28-39

4. Ciaccia, P., Patella, M., Zezula, P.: M-tree: An Efficient Access Method for Similari ty Search in Metric Spaces. In Proc. 23rd Athens Int. Conf. on VLDB, (1997) 426-435
5. Dandamundi, S., Sorenson, P.: An empirical performance comparison of some variations of the k-d tree and BD-tree. Int. J. Comp. Inform. Sci., Vol.14, (1985) 135-159
6. Faloutsos, C.: Gray-codes for partial match and range queries. IEEE Trans. on Software Eng., Vol. 14, (1988) 1381-1393
7. Faloutsos, C., Roseman, S.: Fractals for secondary key retrieval. In Proc. ACM SIGACT-SIGMOD Symp. Principles of Database Systems, (1989) 247-25 2
8. Faloutsos, C.: Searching Multimedia Databases by Content. Kluwer Academic Press, (1996).
9. Freeston, M.: The BANG file: a new kind of grid file. In Proc. ACM SIGMOD, (1987) 260-269
10. Gunther, O., Bilmes, J.: Tree-Based Access Methods for Spatial Databases: I mplementation, and Performance Evaluation. IEEE Trans. on Knowledge and Data Eng., Vol. 3, no. 3, (1991) 342-356
11. Gaede, V., Gunther, O.: Multidimensional Access Methods. ACM Computing Surveys, Vol. 30, no. 2, (1998) 170-231
12. Guttman, A.: R-Trees: A Dynamic Index Structure for Spatial Searching. In Proc. ACM SIGMOD, Boston, (1984) 47-57
13. Jagadish, H.V.: Spatial Search with Polyhedra. In Proc. 6th IEEE Int. Conf. on Data Engineering, (1990) 311-319
14. Kapopoulos, D.G., Hatzopoulos, M.: The G^r_Tree: The Use of Active Regions in G-Trees. In: Eder J., Rozman, I., Welzer T. (eds.): Advances in Databases and Information Systems. Lecture Notes in Computer Science, Vol. 1691. Springer-Verlag, (1999) 141-155
15. Katayama, N., Satoh, S.: The SR-tree: An Index Structure for High -Dimensional Nearest Neighbor Queries. In Proc. ACM SIGMOD, (1997) 369-380
16. Kumar, A.: G-Tree: A New Data Structure for Organizing Multidimensional Data. IEEE, Trans. on Knowledge and Data Eng., Vol. 6, no. 2 (1994) 341-347
17. Lin, K.I., Jagadish, H.V., Faloutsos, C.: The TV-Tree: An Index Structure for High - Dimensional Data. In VLDB Journal, Vol.3, no.4, (1994) 517-542
18. Nievergelt, J., Hintenberger, H., Sevcik, K.C.: The Grid File: an adaptable, symmetric multikey file structure. ACM Trans. Database Syst., Vol. 9, no. 1, (1984) 38-71
19. Orenstein, J., Merrett, T.: A class of data structures for associative searching. In Proc. ACM SIGACT-SIGMOD Symp. Principles of Database Systems, (1984) 181-190
20. Sakurai, Y., Yoshikawa, M., Uemura S., Kojima, H.: The A-tree: An Inde x Structure for High Dimensional Spaces Using Relative Approximation. In Proc. 26th Cairo Int. Conf. on VLDB, (2000) 516-526
21. Samet, H.: The Design and Analysis of Spatial Data Structures. Addison-Wesley, MA (1990)
22. Samet, H.: Spatial Databases. In Proc. 23rd Athens Int. Conf. on VLDB, (1997) 63-129
23. Sellis, T., Roussopoulos, N., Faloutsos, C.: The R^+-tree: A Dynamic Index for Multid imensional Objects. In Proc. 13th Brighton Int. Conf. on VLDB, (1987) 507-518
24. Manolopoulos, Y., Theodoridis, Y., Tsotras, V. J.: Advanced Database Indexing. Kluwer Academic Publishers, Boston (1999)
25. Traina, C., Traina, A., Seeger, B., Faloutsos, C.: Slim-Trees: High Performance Metric Trees Minimizing Overlap Between Nodes. In Proc. EDBT (Extending Database Technology), Konstanz, Germany, (2000)
26. White, D., Jain, R.: Similarity Indexing with the SS-tree. In Proc. 12th ICDE, (1996) 516-523

Evaluation of Join Strategies for Distributed Mediation

Vanja Josifovski*, Timour Katchaounov, and Tore Risch

Uppsala Database Laboratory, Uppsala University, Sweden,
`vanja@us.ibm.com`, `timour.katchaounov@dis.uu.se`, `tore.risch@dis.uu.se`

Abstract. Three join algorithms are evaluated in an environment with distributed main-memory based mediators and data sources. A streamed ship-out join ships bulks of tuples to a mediator near a data source, followed by post-processing in the client. An extended streamed semi-join in addition builds a main-memory hash index in the client mediator. A ship-in algorithm materializes and joins the data in the client mediator. The first two algorithms are suitable for sources that require parameters to execute a query, as web search engines and computational software, and the last is suitable otherwise. We compare the execution times for obtaining all and the first N tuples, and analyze the percentage time spent in subsystems, varying the network communication speed, bulk size, and data duplicates. The join algorithm leads to orders of magnitude performance difference in different mediation environments.

1 Introduction

Integration of data from sources with varying capabilities has been intensively studied by the database community in the recent decade. The Amos II system [8, 9,17] uses the *wrapper-mediator* paradigm to integrate data from several sources. One of the salient features of Amos II is a distributed architecture where a number of interconnected *mediator servers* cooperate in providing the users and the applications with the required view of the data in the sources. We believe that a distributed mediator architecture is needed because it is unrealistic to assume that a single mediator server can be deployed in an enterprise composed of multiple organizational units. When many mediator servers become available on the network, composability will be required for designing new distributed mediator servers in terms of the existing ones, thus reusing mediation specifications. Multiple mediators will also alleviate the performance bottleneck problems that appear when all the queries are handled by a single mediator.

Having some of the basic assumptions different from the classical database systems, query processing in a distributed mediator system requires some novel strategies and solutions. One of the major reason for this is the different cost model in this environment. The I/O and CPU costs used in the traditional query optimization [14] are largely insignificant here compared to the cost of accessing

* Current address: IBM Almaden Research Center, San Jose, CA 95120, USA

data in external sources. While new cost models have been developed for use in mediator frameworks with centralized architecture [18], no experimental results are reported using a distributed mediator framework. In this work we quantify empirically the relations among the different costs in a wrapper-mediator environment, as for example, the network cost and the data source access costs.

Traditional data integration systems [11,16] send all data to the mediator for joining. Such 'ship-in' methods do not allow for integration of 'non-database' data sources that require some input, since it is not possible to ship the programming logic from these systems into the mediator. Also they are not good for top-N queries where only a first few tuples are retrieved.

Three join algorithms for a distributed mediation environment are presented and analyzed. An outer collection, generated as an intermediate result of a previous computation, is joined with an inner collection produced from a data source. Two *ship-out* algorithms ship data *toward* the sources. In these algorithms, intermediate result tuples are shipped to the sources where they are used as parameters to precompiled query fragments (subqueries or function calls) of the original query. The first algorithm is an order-preserving semi-join which is suitable when there are no duplicates in the outer collection. The second algorithm uses a temporary hash index of possibly limited size to reduce the number of accesses to the data sources. It is suitable when there are duplicates in the outer collection. Both ship-out algorithms are streamed [6] and the data is shipped between the mediator servers in bulks that contain several tuples to avoid the message set-up overhead. Finally, for comparison, a ship-in algorithm is analyzed, which is suitable when the sources cannot accept parameterized queries and when the data retrieved from the sources is small enough to be stored in a temporary main-memory index in the mediator.

The algorithms are evaluated in an environment with an ODBC data source and a mediator server running on Windows NT platforms, connected by ISDN and LAN. Substantial performance gains were measured (up to factor 100) when using our framework over an ISDN connection to access a relational database server, as compared to accessing the relational database with ODBC directly from the client, since bulk oriented join processing between the mediators minimizes ISDN message traffic and eliminates all expensive remote ODBC calls.

2 Background

As a platform for the work in this paper we use the Amos II mediator database system [8,9,17]. The core of Amos II is an open light-weight and extensible DBMS. It is a distributed mediator system where both the mediators and wrappers are fully functional Amos II servers, communicating over the Internet. For good performance, and since most the data reside in the data sources, each Amos II server is designed as a main-memory DBMS.

Some of the Amos II servers can be configured to wrap different kinds of data sources, e.g. ODBC compliant relational databases [4] or XML files [12]. Other servers reconcile conflicts and overlaps between similar real-world entities modeled differently in different data sources, using the *mediation primitives* [8, 9,17] of the query language AmosQL .

Users and applications can pose OO queries to any Amos II server. We call the server(s) to which application queries are posed *client mediator(s)* for those queries. The other Amos II servers involved in answering a query are called *mediator servers.* The mediator servers may run on separate workstations and provide data integration, wrapping, and abstraction services through which different views are presented in different mediators. For example, in a mobile environment a portable computer could have a client mediator that integrates data represented by several mediator servers on a company LAN. A mediator server can have different types of data sources attached and access a number of other mediator servers.

The AmosQL query below contains a join and selection over the table A at the source $DB1$, and B at $DB2$, based on values of functions fa and fb:

```
select res(b)
from A@DB1 a, B@DB2 b
where fa(a) = fb(b);
```

The query is issued in a client mediator over data that can be either directly stored in *DB1* and *DB2* or, if these are Amos II servers, retrieved from wrapped data sources. Strategies to execute this equi-join will be the focus of this paper.

The queries are rewritten by the optimizer to eliminate redundant computations. After the rewrites, queries operating over data outside the mediator are decomposed into distributed *query fragments*, executed in different Amos II servers and data sources. The decomposition uses heuristic and dynamic programming strategies in three stages [10]: query fragment generation, fragment placement and fragment scheduling. Each Amos II server uses a single-site *cost-based optimizer* to generate optimized execution plans for the query fragments. The fragments for other types of data sources are handled by the mediator if the source has no query processing capabilities, or by the source otherwise.

3 Algorithm Descriptions

While a naive data source interface provides only *execute* functionality for queries, Amos II also provides *bulked ship-out and execute* functionality where a remote Amos II server accepts and store tuples locally in main-memory, and then executes a query fragment using them as an input. When joining directly to a data source, the communication is directly with it and the processing is one tuple at the time for the ship-out algorithms, assuming that storing bulks of the intermediate results is not possible in data sources because of their autonomy.

3.1 Ship-Out Join Algorithms

In general, the ship-out algorithms can be described with the following steps:

1. preprocess and prepare the input collection for shipping
2. ship the input collection to a remote site
3. execute the query fragment over the collection at the remote site
4. return result of query fragment execution to the coordinating mediator
5. assemble the result collection to be emitted from the join

Steps 1, 4 and 5 are executed locally, while 2 and 3 are performed at another Amos II server by its join request handler.

The input collection is a table where some columns are used as parameters to the remote query fragment; other columns are passed through to the later post-processing in the mediator, or are assembled as parts of the query result.

A straight-forward implementation of a ship-out equi-join operator would ship the whole input bulk to the remote site, execute the remote query fragment on the bulk appending its result to the input, and then ship this result back. The first improvement of the naive strategy we propose is the *project-concat algorithm* (PCA) in Fig. 1[1]. It improves the naive strategy by the following two data transformations based on the semi-join algorithm [2]:

– The input bulk is projected over the data columns that are actually used in the remote query fragment, before shipping them there.
– After the query fragment is executed the result shipped back to the mediator contains only the relevant columns from the query fragment result.

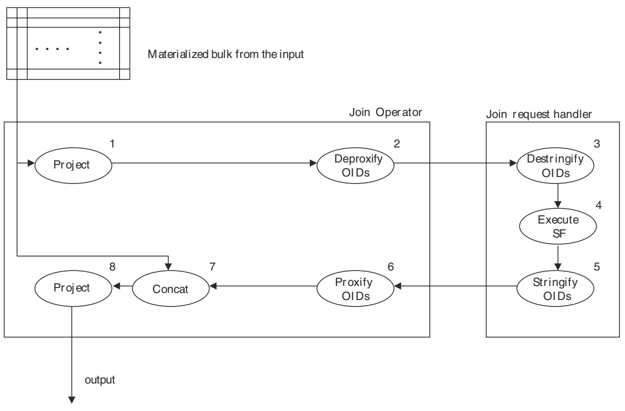


Fig. 1. Project-concat ship-out algorithm

[1] Amos II is object-oriented and steps 2, 3, 5, and 6 handle object identifier (OID) conversions, which are not further elaborated here.

Table 1. Example execution of equi-join using the project-concat algorithm

	r	**va**
b1	“T”	5
	“V”	4
	“K”	5
	“M”	3
b2	“L”	5
	“M”	2
	“G”	4
	“Y”	4

$\xrightarrow{\pi_{KS}}$

	va
b1	5
	4
	5
	3
b2	5
	2
	4
	4

$\xrightarrow{to\ DB1} \ldots$

	tmp
b1	false
	false
	false
	true
b2	false
	true
	false
	false

$\xleftarrow{from\ DB1} \ldots$

	r	**va**
b1	“M”	3
b2	“M”	2

$\xleftarrow{concat}$

	r
b1	“M”
b2	“M”

$\xleftarrow{\pi_{RS}}$

The difference between PCA and the classical semi-join is in the use of order for matching the tuples from the joined collections.

The result of the join is assembled by a simple concatenation of the input and the result shipped back from the remote Amos II mediator or data source. Since the operations are order preserving, concatenation can be used instead of a more expensive join.

Table 1 illustrates an execution of PCA between the results of query fragments $QF1$ executed at $DB1$ and $QF2$ executed at $DB2$. The input is a collection of tuples with columns va and r produced by the execution of the fragment $QF2$, and a collection of tuples produced by the execution of $QF1$ containing va values and keys of table tB. The fragments are joined over va and the result is represented by column r. Since there are no result columns that are shipped back from DB1 to DB2, a boolean value is used to identify if the tuples produced by $QF1$ have a matching va value in the tuples produced by $QF2$. We assume that the fragment at $DB1$ produces the following table:

va	
tB	**va**
ib_1	4
ib_2	5
ib_1	6

where ib_k denotes a key of tB. The example illustrates the execution over 2 bulks of size 4, named in the example as $b1$ and $b2$. In the example, first the projection strips the r values from the input bulks since they are not used in the join. Next, the bulks are shipped to $DB1$ where the query fragment $QF1$ is executed. The resulting set of boolean values is shipped back to the mediator. The concatenation shown in the example is a special case where the executed function does not return any data used later in the query processing. In this case, the concatenation of the returned boolean values and the input tuples actually filters the tuples for which the result is $true$. The final projection removes the va values to form the requested result.

The PCA has the advantage of improving the naive implementation, while preserving the simplicity of the processing. All operations have constant complexity per data item and therefore cheap to perform. Nevertheless, it is inefficient when there is a large percentage of duplicates in the input bulk(s), an expensive query fragment, and/or expensive communication between the servers involved.

The traditional *semi-join* algorithm (SJA) [2] improves the performance of the PCA when duplicates are involved. After projecting the input bulk over the columns used as input to the remote query fragment, SJA performs duplicate removal before shipping the data. When there is a large percentage of duplicates within the bulks, this reduces both the size of the shipped data and the number of executions of the remote query fragment. The result of the query fragment execution is shipped back to the calling server where, as in the previous algorithm, the shipped tuples are concatenated to the result of the query fragment invocation. Next, an equi-join is performed over the input bulk and the result of the concatenation. Here, because of the duplicate removal it is not possible to match the tuples by their rank in the bulk.

The SJA benefits from avoiding shipping duplicate entries over the network and executing the query fragment for them, but only for duplicates within a single bulk and with the added costs of the two additional phases of duplicate removal and equi-join.

To avoid duplicates over different bulks, the algorithm in Fig. 2, SJMA (semi-join with materialized index algorithm) extends SJA by saving the index built up for the bulks of the outer collection between executions for different bulks. The shipped data is passed through an additional anti-join over the set already pruned from duplicates and the temporary index. If a tuple is in the index, it has already been processed in some of the previous bulks. The remaining tuples are shipped to the remote site for query fragment execution as before. Next, new entries are added to the index from the returned result. Finally, a join between the input bulk and the index is performed as in the SJA. A comparative execution of SJMA in the same scenario as for the PCA example is presented in Table 2. Here, the second bulk is reduced to one tuple before shipping to $DB1$, since the anti-join eliminates the two tuples present in the first bulk.

The size of the index in SJMA is proportional to the number of distinct tuples in the outer collection. The algorithm can be used as a filter even in the case when the whole index is too big to fit in the memory. When the memory limit is reached, new entries replace old entries using some replacement criteria.

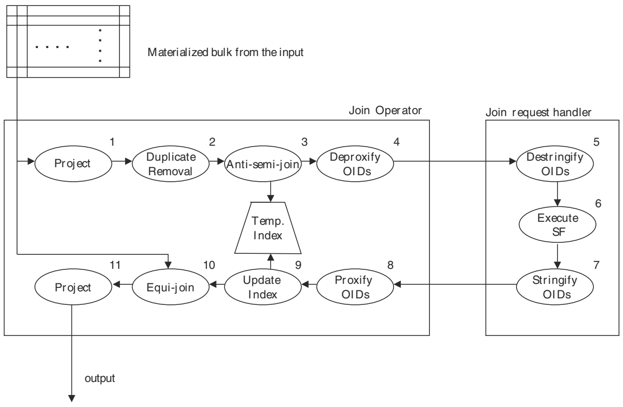


Fig. 2. Streamed semi-join with a temporary index

Table 2. Example execution of the semi-join with materialized index algorithm

	r	**va**
b1	"T"	5
	"V"	4
	"K"	5
	"M"	3
b2	"L"	5
	"M"	2
	"G"	4
	"Y"	4

$\xrightarrow{\pi_{KS}}$

	va
b1	5
	4
	5
	3
b2	5
	2
	4
	4

$\xrightarrow{dup.\ rem.}$

	va
b1	5
	4
	3
b2	5
	2
	4

$\xrightarrow{anti\ semi\ join}$

	va
b1	5
	4
	3
b2	2

$\xrightarrow{to\ DB1} \ldots$

	r
b1	"M"
b2	"M"

$\xleftarrow{\pi_{RS}}$

	r	**va**
b1	"M"	3
b2	"M"	2

$\xleftarrow{equi\text{-}join}$ $\triangle$ $\xleftarrow{index\ update}$

	tmp
b1	false
	false
	true
b2	true

$\xleftarrow{from\ DB1} \ldots$

SJMA does not add substantially to the cost of the SJA, while it offers the possibility for performance improvements. In fact, it reduces to the SJA in the case when the whole input is contained in only one bulk.

3.2 Ship-In Join Method

Unlike the previous two algorithms where the remote query fragment is executed using parameters from the tuples of the intermediate result, with the ship-in join method no intermediate result is shipped to the remote site. Consequently, the query fragment is executed without parameters. This has two effects:

– Since the remote query fragment is executed once only, it may reduce the number of accesses to the data source.

– The result size may increase since instead of a semi-join of the query fragment result and the intermediate result, the whole query fragment result is sent to the client to be joined there.

While the reduction of the data source accesses may improve the performance, the increased volume of the data shipped and stored in the mediator are the possible performance disadvantages of this algorithm. The algorithm is inapplicable when the query fragment result is too big for the mediator resources. This is also the case when the query fragment contains predicates representing methods/programs in the data source that require parameters to be supplied from the mediator. With the ship-out method, when there are sufficient resources, the materialized index can persist between the execution of the algorithm for different bulks, reducing further the query processing time. This case corresponds to hash join algorithms where an index is built for the inner relation.

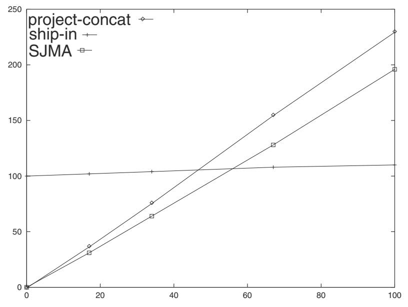
4 Performance Measurements

In the two scenarios used in the experiments the data source was an ODBC data source. We performed experiments using both Microsoft Access ODBC and IBM DB2 ODBC drivers with no significant differences in conclusions. Where not specifically indicated, the measurements use the Access ODBC driver.

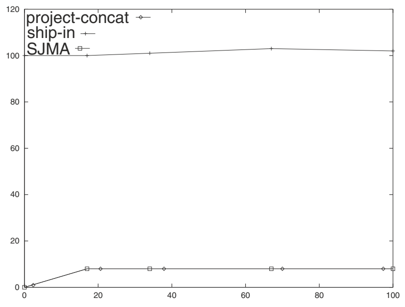
In the first scenario, we deployed an Amos II server at the same workstation as the source. This server wrapped the source and exported it to the client mediator running on another Windows NT workstation. We present test results using this scenario and two different network connection speeds between the workstations: a 115Kb ISDN connection over the public telephone network in Sweden; and a 100Mb departmental LAN. We also varied the speed of the workstation that hosted the client mediator. In one experiment we used a 233 MHz, 32Mb RAM PC, and in the other a 600 MHz, 256Mb RAM PC. In the second scenario the data source was accessed directly from the client mediator through the ISDN network connection using DB2's ODBC interface. In this case the joins are executed one tuple at a time. We also compared the effects of different bulks sizes on the query execution time.

The inner collection is obtained from a table stored in the ODBC data source. The table consisted of three columns: an integer primary key ID, and two textual columns A and B of fixed length strings with sizes 10 and 250. The outer collection was stored in the client mediator, where it simulated an intermediate result. During the execution, the outer collection is bulked and streamed into the join algorithm one bulk at the time. Both the outer and inner collection had the same attributes.

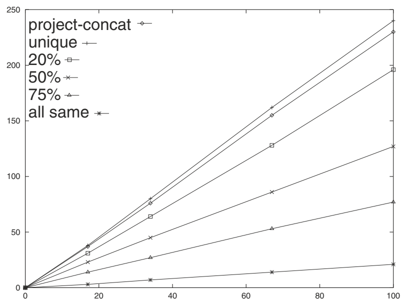
Figure 3 shows the results of the execution of the three join algorithms from the previous section using a 233 MHz Windows NT workstation as a client and an ISDN connection to the server computer. The X axes in the graphs show the sizes of the outer collection in percentage of the size of the inner that always contains 30000 tuples; the Y axes marks query execution times in seconds. The outer collection is scaled from 17% to same the size as the inner. In these experiments



(a) Whole result



(b) First 1024 tuples



(c) SJMA with different percentage duplicates

Fig. 3. Execution times when varying the outer collection size, ISDN, 233 MHz PC

the outer collection contained 20% duplicates. Each tuple of the outer matches exactly one tuple of the inner. The graph on the left compares the execution times for a complete evaluation of the join operation. The graph in the middle compares the times to emit the first 1024 tuples. This coincides with he bulk size used to execute the query. The graph on the right compares the SJMA with PC for different percentages of duplicates in join columns of the collections.

We first analyze the execution times for the complete join operation. Since the inner collection has constant size, the time spend in the Amos II server of the inner and the network time are constant for the execution of the ship-in algorithm. The only increase of execution time is noted in the client: from 8 seconds for a 5000 tuple outer collection, to 16 seconds for a 30000 tuple outer collection. This is due to the increase of the number of index searches. Nevertheless, this increase in negligible in comparison to the total query execution time.

Table 3. Query execution time distribution, ISDN, 233 MHz PC

	Time distribution			
	Client	Server	Source Access	Net.
Ship-in	10%	1%	4%	85%
Ship-out, PC	5%	3%	43%	49%
Ship-out, SJMA	7%	3%	42%	48%

The ship-out algorithms show performance that is linear to the size of the inner collection, outperforming the ship-in algorithm until the outer is about 50% of the inner. SJMA performs better than the PC algorithm. Figure 3c compares the algorithms for different percentages of duplicates. The PC algorithm performs exactly the same, regardless of the data distribution. SJMA improves as the number of duplicates of the join columns increases. Note that even without duplicates, the performance difference of these two algorithms is small. This shows that in main-memory based mediator systems, the penalty of the additional steps of the SJMA is low.

Table 3 shows the portions of the time spent in the individual system components. The data source access time includes the time spent in the ODBC interface and the data source. The main portion of the execution of the ship-in algorithm executed over ISDN is spent on shipping the inner to the client side, which was consistently around 85% of the query execution time. We can also note that, due to the main-memory architecture of Amos II, the index build time in the client is relatively small, around 5% of the whole execution time. The first tuple is not emitted until the index for the inner is finished, which is after 95% of the processing time. This makes this algorithm unsuitable for top-N queries.

The ship-out algorithms spend less time on the network, but more in accessing the data source. They also emit the first tuple much faster than the ship-in algorithm (Fig. 3b). The experiments show here the time to emit the first 1024 tuples. When the bulking factor is less than 10, the first tuple is emitted after less than a millisecond. Furthermore, the bulking factor also determines the smoothness of the flow of the results. Smaller bulking factor will allow smoother flow of the results to the application.

Table 4 compares the effect of the distributed Amos II architecture for the ship-out algorithms. First we used SJMA to access a remote IBM DB2 data

Table 4. Direct access to an ODBC source and through Amos II servers

	outer/inner			
Inner size/outer size	17%	33%	66%	100%
through Amos II, all tuples	58	115	245	358
ODBC direct, all tuples	2769	5059	8552	12799
through Amos II, B=1024, first tuple	14	15	15	15
through Amos II, B=1, first tuple	0.7	0.72	0.68	0.71
ODBC direct, first tuple	1.1	0.9	1.2	1.04

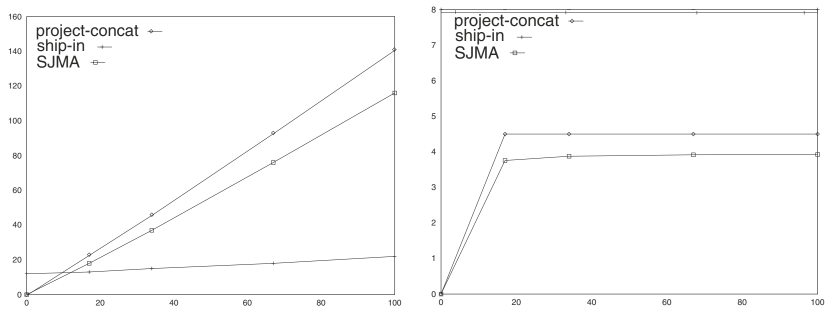
Table 5. Query execution time distribution, 100Mb LAN, 233MHz PC

	Time distribution			
	Client	Server	Data source acc.	Net.
Ship-in	67%	7%	22%	4%
Ship-out, PC	8%	5%	86%	1%
Ship-out, SJMA	12%	5%	82%	1%

source using DB2's ODBC interface over an ISDN connection. Due to the autonomy of the data sources we assume that it is not feasible to materialize intermediate results in the sources. Even if this was possible, due to the disk based nature of the DBMS, we could not expect a comparable execution time as with the main-memory storage used in Amos II. Therefore the join must be performed one tuple at a time over the remote ODBC. However, when the source is accessed through an Amos II server located on the same computer as the source, the join between the client and server mediators is executed in a bulked manner, using only the local ODBC connection between the server mediator and the source, leading to performance improvements of orders of magnitude.

The time to emit the first tuple when the bulking factor is 1024 is notably greater when the processing is done through an Amos II server. This actually represents how long it takes to emit the first 1024 tuples. If fewer tuples are required, a smaller bulking factor leads to better performance for the top-N queries when an intermediate Amos II server is used. Even when the bulking factor is 1 we can note that the use of an intermediate Amos II yields better performance than accessing the source directly, due to communication protocol differences. To achieve the best performance, the bulking factor should match the number of tuples required immediately.

Figure 4 and Table 5 illustrates join execution time on the same client computer connected with a 100Mb fast LAN to the data source. We can note that the curves have similar shapes, while the scale is different. The network cost is eliminated for almost all of the algorithms. In this executions most of the time is spend in the data source (parameterized and unparameterized query execution) and in the client for the ship-in algorithm (index build-up and join). We can also note that when the whole join result is required the ship-in algorithm outperforms the ship-out in almost all the cases. When the first-N tuples are required, however, the ship-out algorithms are more efficient. For the first 1024 result tuples the difference is about 50%. If the number of requested result tuples is smaller, the difference can be a couple of orders of magnitude. We have also varied the client computer from a workstation to a notebook. We noted that the return time for the first tuple is almost constant for the ship-out algorithms regardless of the power of the client computer. This can be explained by the fact that in the case of ship-out algorithms, the server uses the larger share of the workload than with the ship-in algorithms.



(a) Whole result (b) First 1024 tuples

Fig. 4. Join execution times for different outer collection sizes in percentage of the inner size, 100Mb LAN, client 233MHz PC

5 Related Work

The System R* project [14] is one of the first distributed database prototypes. In System R*, both ship-in and ship-out strategies are examined. In [15] a disk-based ship-in strategy (named ship-whole) is implemented with a disk based b-tree index. This type of implementation leads to considerably different results where the ship-out method always outperforms ship-in.

Disk-based semi-join algorithms are described in [1,2,5,14]. A sort-merge join, bloom filter semi-join, and sort-based semi-join are evaluated in [15] for a distributed database environment. A bloom filter phase can be added to the ship-out algorithms described in this paper. Nevertheless, this would incur additional query processing overhead and possibly shipping of some extra tuples of the inner collection. Bloom filter strategies cannot be used with sources that cannot enumerate the extent of the inner collection.

Most of the mediator frameworks reported in the literature (e.g. [7,16,19]) propose centralized query compilation and execution coordination. In [3] it is indicated that a distributed mediation framework is a promising research direction, but to the extent of our knowledge the results in this area are sketchy without experimental support. The protocols for execution of joins between data in different sources are in most cases based on retrieving the data from the sources and assembling the results in the mediator [16,19]. In the DIOM project [13], a distributed mediator system is presented where the query execution is performed in two phases: subquery execution and result assembly. The dataflow is only from the sources to the mediator.

The Garlic mediator system [7] is the only mediator system known to us that supports ship-out join strategies. The *bind join* in Garlic sends parameters to the sources as single tuples of values. In Amos II the data sources are also accessed one tuple at the time, but the distributed architecture allows for using bulked protocols over high latency lines between Amos II servers to avoid most of the processing cost. A Garlic wrapper that has two components, one local and one remote, could achieve the benefits of the approach described in the paper. Finally, join methods where bulk shipping is combined with hashing are not applied in Garlic.

6 Summary and Conclusions

An efficient data integration system needs to be able to adapt to different environments by using different algorithms. The algorithms presented in this paper allow for balancing the workload between the client and the server, and for different network use patterns that give wide range of options over different hardware platforms.

The experimental results showed that for a complete query answer the ship-in algorithms generally outperform the ship-out algorithms over fast networks. Over slow networks and with very slow sources, the ship-out algorithms can give orders of magnitude better performance than ship-in since ODBC over TCP/IP calls are executed one tuple at a time while bulks of tuples are shipped between the distributed mediators. For top-N queries where N is considerably smaller than the result size, the ship-out algorithms with bulking factor N give the best performance over all the range of hardware and network connections used in the experiments. These outperform the ship-in algorithms by a few orders of magnitude. Although the bulking factor greater than 1 provides benefits, too large bulk sizes lead to reduced query execution efficiency.

In our environment, where the index operations are main-memory based and relatively cheap, the penalty of SJMA (the Semi-Join with Materialized index Algorithm) is small and it always performs nearly as well, or better than PCA (the Project-Concat Algorithm). Nevertheless, PCA uses less memory and could be much more efficient in memory-limited mediators. A compromise between these two algorithm is the SJMA with a limited size temporary index that degenerates to a SJA when the temporary index size is 0. Finally, if simplicity of implementation is considered the PCA is the algorithm of choice.

Placing an mediator server close to the source allows for bulked execution of the protocols that might change the query execution time by orders of magnitude, especially in networks with high latency. In cases when the sources lack filtering capability, the mediator server can also locally filter the query fragment result and reduce the communication cost even more.

A topic of our current work is a strategy to dynamically select between the proposed algorithms during run-time. Statistics collected during the execution can be used to determine if the default choice was the best one. Another open issue is a method to determine the optimal bulking factor in a multi join query, by taking in account the tuple sizes, join selectivities and the buffer pool size.

References

1. P. Apers, A. Hevner, and S. Yao: Optimization Algorithms for Distributed Queries. *IEEE Transactions on Software Engineering*, 9(1), 57-68, 1983
2. P. Bernstein and D. Chiu: Using Semi-joins to Solve Relational Queries. *Journal of ACM* 28(1), 25-40, 1981
3. W. Du and M. Shan: Query Processing in Pegasus, In O. Bukhres and A. Elmagarmid (eds.): *Object-Oriented Multidatabase Systems*. Pretince Hall, 449-471, 1996.
4. G. Fahl and T. Risch: Query Processing over Object Views of Relational Data. *The VLDB Journal*, Springer, 6(4), 261-281, 1997.
5. P. Bernstein, N. Goodman, E. Wong, C. Reeve, J. Rothnie Jr.: Query Processing in a System for Distributed Databases (SDD-1). *ACM Transactions on Database Systems (TODS)*, 6(4), 602-625, 1981
6. G. Graefe and W. J. MCKenna: The Volcano Optimizer Generator: Extensibility and Efficient Search. *12th Data Engineering Conf. (ICDE'93)*, 209-218, 1993.
7. L. Haas, D. Kossmann, E.L. Wimmers, J. Yang: Optimizing Queries across Diverse Data Sources. *23th Intl. Conf. on Very Large Databases (VLDB'97)*, 276-285, 1997
8. V.Josifovski and T.Risch: Functional Query Optimization over Object-Oriented Views for Data Integration. *Intelligent Information Systems (JIIS)* 12(2-3), Kluwer, 165-190, 1999.
9. V.Josifovski and T.Risch: Integrating Heterogeneous Overlapping Databases through Object-Oriented Transformations. *25th Intl. Conf. on Very Large Databases (VLDB'99)*, 435-446, 1999.
10. V. Josifovski and T. Risch: Query Decomposition for a Distributed Object-Oriented Mediator System. To appear in *J. of Distribued and Parallel Databases*, Kluwer, 2001.
11. E-P. Lim, S-Y. Hwang, J. Srivastava, D. Clements, and M. Ganesh: Myriad: Design and Implementation of a Federated Database System. *Software - Practice and Experience*, Vol. 25(5), 553-562, John Wiley & Sons, May 1995.
12. H. Lin, T. Risch and T. Katchanounov: Adaptive data mediation over XML data. To appear in *J. of Applied System Studies (JASS)*, Cambridge International Science Publishing, 2001.
13. L. Liu and Calton Pu: An Adaptive Object-Oriented Approach to Integration and Access of Heterogeneous Information Sources. *Journal of Distributed and Parallel Databases* 5(2), 167-205, Kluwer Academic Pulishers, The Netherlands, 1997.
14. G. Lohman, C. Mohan, L. Haas, D. Daniels, B. Lindsay, P. Selinger and P. Wilms: Query Processing in System R*. In W. Kim, D. Reiner, D. Batory (eds.): *Query Processing in Database Systems*, Springer-Verlag, 1985.
15. L. Mackert and G. Lohman: R* Optimizer Validation and Performance Evaluation for Distributed Queries. In M. Stonebraker (ed.): *Readings in Database Systems*, Morgan-Kaufmann, CA, 1988
16. F. Ozcan, S. Nural, P. Koksal, C. Evrendilek, and A. Dogac: Dynamic Query Optimization in Multidatabases. *IEEE Data Engineering Bulletin*, 20(3), 38-45, 1997.
17. T. Risch and V. Josifovski: Distributed Data Integration by Object-Oriented Mediator Servers. To appear in *Concurrency - Practice and Experience J.*, John Wiley & Sons, 2001.

18. M. Roth, F. Ozcan and L. Haas: Cost Models DO MAtter: Providing Cost Information for Diverse Data Sources in Fededrated System. *25th Intl. Conf. on Very Large Databases (VLDB99)*, 599-610, 1999.
19. A. Tomasic, L. Raschid and P. Valduriez: Scaling Access to Heterogeneous Data Sources with DISCO. *IEEE Transactions in Knowledge and Data Engineering*, 10(5), 808-823, 1998

An RMM-Based Methodology for Hypermedia Presentation Design

Flavius Frasincar, Geert Jan Houben, and Richard Vdovjak

Eindhoven University of Technology
Eindhoven, The Netherlands
{flaviusf,houben,richardv}@win.tue.nl

Abstract. Due to the rapid growth of the Web, there is an increasing need for methodologies that support the design of Web-based Information Systems (WIS). After investigating the application of existing hypermedia design methodologies in the context of automated hypermedia presentation design we propose a specification framework for this context. The framework considers the possibility of dynamically gathering information from a collection of structured, but also possibly heterogeneous sources (relational or object-oriented databases, XML repositories etc.). The methodology associated with the framework shows two levels of abstraction: the logical level, and the presentation level. At the logical level the application diagram captures the design of slices, thus specifying the content related grouping of data elements and their relationships. At the presentation level, the presentation diagram bridges the logical level and the actual implementation by specifying how the design of slices is translated into hypermedia mechanisms, e.g. hyperlinks.

1 Introduction

From its introduction in the early 90's the World Wide Web (WWW) is in a continuous development. Its rapid expansion results in an increasing number of Web-based Information Systems (WIS) [3] being developed, especially with sources that contain frequently changing information such as databases (relational or object-oriented databases, XML repositories etc.). This leads to the fact that there is a higher need to automate, at least partially, the design process of hypermedia presentations as used in a WIS. Although there are methodologies like Relationship Management Methodology (RMM) [1] and Object Oriented Hypermedia Design Methodology (OOHDM) [2], these methodologies have been originally developed for a manual hypermedia design process, they are not particularly well-suited in the context of automated hypermedia design.

RMM focuses on highly structured applications with high information volatility. Moreover, it provides guidelines that can facilitate the automated design process of hypermedia applications. RMM is based on the popular Entity Relationship (E-R) model.

These characteristics are significant for our target application area where sources contain dynamical information. RMM has a specification at the logical level, which groups presentation issues (e.g. navigational links) with semantical issues (e.g. slice attributes). Besides, it lacks a proper specification at the presentation level. In this paper we argue that the separation in two distinct levels is useful and we address them both in the context of the proposed framework.

The results of this research will be exploited in the HERA [12] project that investigates software architectures for the generation of presentations for (semi-structured) data retrieved from heterogeneous data sources. In order to experiment with the proposed methodology a prototype has been built. The prototype uses XML to store the data and XSL to specify transformations between consequent steps. Similar model-driven Web systems inspired by Hypermedia Design Methodology (HDM) [7] were developed in Autoweb [6] (uses relational databases for data storage) and WebML [5] (uses XML for data storage) projects.

Part of this work is also embedded in the Dynamo project, in which Eindhoven University of Technology collaborates with Philips and CWI, both in the Netherlands. The Dynamo project targets the (semi-)automatic generation or design of multimedia presentations. Characteristic for the generation process is the support for adaptation.

As results from [4], [10] show, adaptation can be specified by distinguishing the domain model, the user model, and the adaptation model. The last model describes the actual adaptation being performed (based on the assumed knowledge of the user inferred from the user model). In terms of this project it means that conditions are specified on the existence of objects that are part of the presentation.

2 RMM

RMM [1], [11], [13], [14], [15] proposes a sequence of steps that have to be followed in the design process. For each step there are design guidelines for the designer or builder of the application. Figure 1 depicts the RMM steps that are reused in the proposed framework.

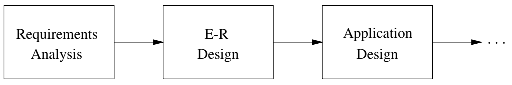


Fig. 1. RMM design methodology

The first design step is to represent the application domain using traditional E-R modeling techniques.

In the second step, the E-R model is enriched with the concepts of m-slices and relationships to build a diagram that describes the application. M-slices

[13] are created by grouping together attributes from the E-R diagram; this can be done hierarchically using previously defined m-slices: the ‘m’ comes from ‘Matreyshka’, the nested Russian doll, suggesting the possibility to nest slices in one another. In the rest of the paper we will use the term slice instead of m-slice. The most simple type of m-slice is an attribute ensuring thus the uniformity of the hierarchical slice model.

3 Methodology

We developed a specification framework based on RMM to support the automation of a hypermedia presentation design. Figure 2 illustrates the methodology related to this framework.

Although requirements analysis, E-R diagram (domain analysis), and implementation are important phases in the software engineering life cycle they are beyond the scope of this paper.

The arrows in Fig. 2 show the sequence of steps in the process[1]. Our methodology focuses on the design steps shown in the shaded area of Fig. 2: platform independent application design, platform dependent application design, and presentation design.

The artifacts of each design step in the order in which they appear in the methodology are:

- E-R diagram
- platform independent application diagram
- (platform dependent) application diagram
- presentation diagram

A platform independent application diagram presents common information of application diagrams that is going to be displayed regardless of the platform. A platform dependent application (also referred as application diagram) takes into consideration a particular hypermedia platform.

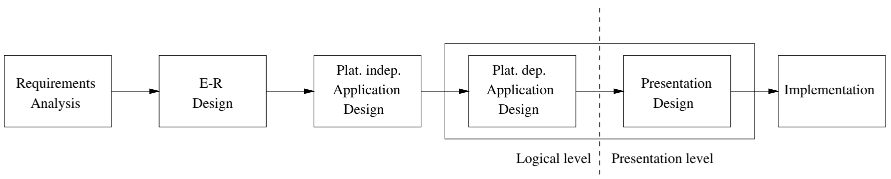


Fig. 2. Design methodology

In the methodology we distinguish two levels, the logical level and the presentation level, based on the ‘separation of concerns’ principle.

[1] It is possible to have feedback loops in the process.

3.1 Logical Level

At the logical level the methodology is concerned with providing data and relationships/access structures of the information to be presented. Related information is grouped together into meaningful presentation units (slices). Slice relationships provide a means to access information belonging to different units. The logical level is based on the E-R model, in the sense that it exploits the entity structure (attributes) and the relationships among them.

Relationships derived (based on relation's transitivity) from the E-R model are taken into account to minimize the number of slice relationships that need to be traversed in order to reach a meaningful item related to a presentation unit.

The logical level bridges the data source level and the presentation level. The designer is concerned with mapping elements from the data sources (at abstract level entities from E-R model) to slices.

Besides structuring information the notion of slice facilitates reuse. A slice can be reused at application level or even among applications if the applications share the same data model. At application level a previously designed slice can be nested in another slice so that the design effort involved in a new slice creation is considerably reduced.

The logical level is expressed in the application diagram which will be described in Sect. 4.

3.2 Presentation Level

The design choices made at the logical level yield a presentation specification in terms of slices. Such a specification however, stays at the logical level and does not describe the presentation in detail. It does not say anything about how presentation elements should be organized with respect to each other or with respect to the screen; it says neither when, nor in which order presentation elements should appear. These kinds of design decisions were usually made ad hoc during the implementation phase, though they clearly do not belong there.

The presentation level bridges the logical level and the actual implementation. Complementary to the logical level, where the designer is concerned with organizing the overall presentation structure and identifying what attributes from entities are to be included in slices, at the presentation level the designer specifies how and when the identified slices should be displayed.

The presentation level should not be considered only as a detailed logical level but rather as a view from a slightly different angle. During this process, slices from the logical level are mapped into regions and design decisions are made to specify a concrete way (navigational, spatial, and temporal) [9] how rather abstract slice relationships will appear in the presentation. In other words the designer specifies in a conceptual way the navigation, layout, and timing of the presentation.

The presentation level is expressed by means of the presentation diagram which will be described in Sect. 5.

4 Application Diagram (AD)

The application diagram is used to model at the logical level the hypermedia application [15]. An application diagram consists of slices and the relationships among them.

In order to illustrate the framework specification an example of a virtual Web site that presents information about a set of real-estate agencies is used.

Figure 3 describes the underlying E-R diagram without attributes. The application domain is simplified in order not to burden the reader with too many application details. In the example we introduce four entities, house agency, house, neighbourhood, and room, and three relationships, expressing that a house is owned by a house agency, it is placed in a certain neighbourhood, and contains several rooms. All relationships are of type one-to-many.

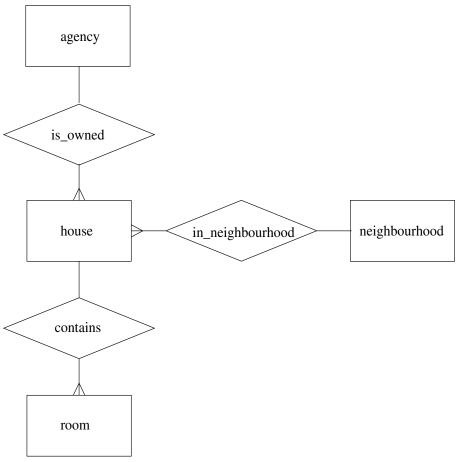


Fig. 3. E-R diagram

4.1 Slices

The application diagram is based on the concept of slice, which groups together attributes and possibly other slices. Each slice belongs to an entity and it can also contain attributes or slices from other entities. Slices provide flexible aggregation and contextual navigation, useful features for intelligent Web navigation.

Slices are linked together with slice relationships. Access structures (e.g. index, tour, indexed guided tour) are used to build complex internal slice structures. Relationships/access structures have associated conditions[2] that show what slice instances are to be connected.

[2] A user adaptation module can use these conditions in order to inhibit or to allow certain links based on the knowledge a user has regarding a certain concept.

The application diagram consists of all the designed slices and provides a global view of the application.

Figure 4 presents a slice (as it is in RMM) which is an element of the larger application diagram.

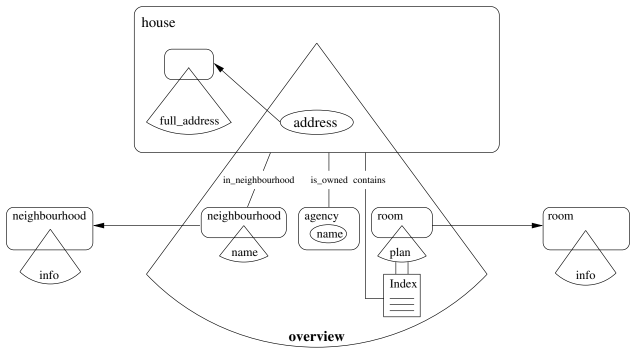


Fig. 4. Slice

4.2 Slice Relationships

Slices are linked together by relationships. A slice relationship associates slices (belonging possibly to different entities) some being sources and others targets. A simple slice relationship connects just two slices while a complex slice relationship connects multiple slices. Each slice relationship can be have conditions (logic predicates) that determine which instances of an entity are to be accessed. Conditions support user and platform adaptation. Figure 5 (left) presents a simple slice relationship.

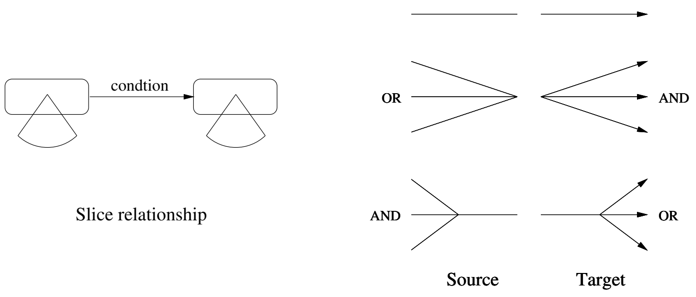


Fig. 5. Slice relationship. Sources and targets for slice relationship

Figure 5 (right) depicts the possible types of sources and targets of a slice relationship. We distinguish the following cases:

– simple source/target
– multi-and source/target, all sources/targets must be triggered/invoked
– multi-or source/target, at least one of the sources/targets is triggered/invoked

One can observe that the proposed notation has as consequence that an 'OR' for sources has the same graphical representation as an 'AND' for targets (and the other way around).

The RMM slice relationship is a navigational relationship (hyperlink) between two slices. In our application diagram a slice relationship represents navigational, spatial, or temporal relationships at a higher level of abstraction. In RMM, slice relationships are always crossing the physical boundaries of a slice. In the proposed framework we allow slice relationships to be embedded in one slice (we call them internal slice relationships). In the case of internal slice relationships the source slice can vanish or can be preserved. Figure 6 presents the associated graphical notation for the two cases.

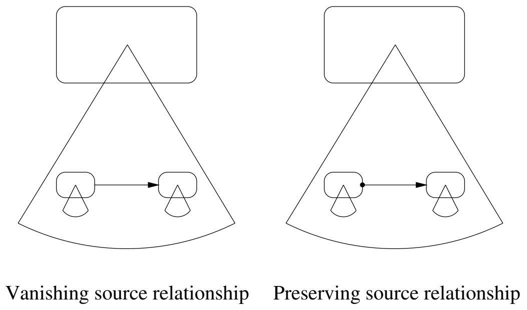


Fig. 6. Vanishing source and preserving source slice relationship

Figure 7 presents a slice diagram that uses the newly introduced concepts of internal relationships and preserving source relationship. The empty slice called 'h_n' is an anchor for two internal relationships (both the anchor and the target belong to the more general 'description' slice). This is an example of relationships that are preserving their source (the bullet anchored on the empty slice). That means that when such a relationship is traversed, the description slice context is kept and the 'h_n' anchor is still present.

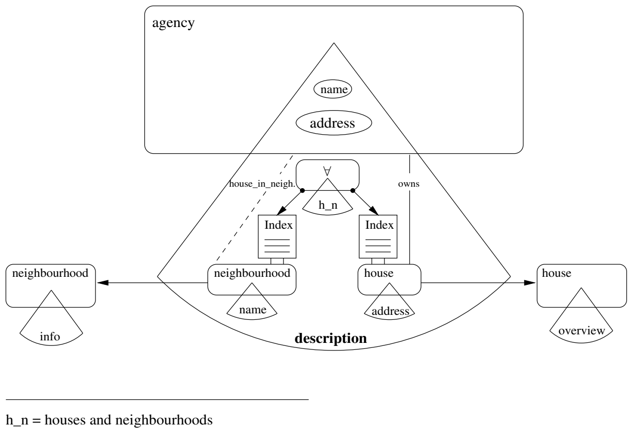


Fig. 7. Slice

5 Presentation Diagram (PD)

Similarly as the logical level is described by means of an application diagram, the presentation level is described by a presentation diagram. A presentation diagram consists of presentation elements called regions and the relationships among them. The presentation diagram specifies how the regions are organized from the navigational, spatial, and temporal point of view. Figure 8 shows an example of a presentation diagram composed from several regions.

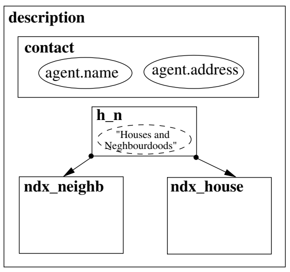


Fig. 8. Presentation diagram

During the process of creating the presentation diagram, the slices introduced at the logical level are mapped to regions and possibly new regions are introduced (it is not always a one-to-one mapping, some slices can be split to more regions and vice versa).

Subsequently, slice relationships from the logical level are materialized with navigational, spatial, or temporal relationships.

5.1 Regions

Regions, together with the relationships that interconnect them, are the main building blocks of a presentation diagram. A region is defined as a collection of attributes and possibly other regions. This recursive definition facilitates nested regions thus providing a means to reuse regions in a component-like manner. Every region is associated with an area where it is displayed. An area is of a rectangular shape having as properties its dimensions and location. When we refer to a region we mean both the region and its associated area.

As mentioned before, a region contains attributes. Attributes are properties that relate concrete values (from a given domain) to the region. Attributes come in two flavors: entity attributes and constant attributes. An entity attribute acquires its value from the entity to which it is mapped and thus it changes its value throughout the presentation. A constant attribute, as its name suggests, does not change its value; the value is determined beforehand and stays constant. Different syntax is used to differentiate the two types. Graphically, entity attributes are depicted as solid line ovals and constant attributes as dashed line ovals as shown in Fig. 8.

Note that though the concept of region is similar to the slice concept, there are some differences:

- A slice is always associated with an entity that owns it, while a region, as a presentation element, is not owned by an entity; it is sufficient that each entity attribute includes the information from which entity it comes from.
- A region has an associated area where it is displayed.

5.2 Region Relationships

All aspects of the abstract concept of slice relationships introduced at the logical level are used also at the presentation level with the difference that the sources and the targets are regions, not slices. Moreover, three instances of this concept are introduced: navigational, temporal, and spatial relationship. Each of them is having a different syntax and different semantics associated with its realization.

Graphically, relationships are depicted as arcs with arrows. Similar to the logical level, an arc starting with a solid circle indicates a persistent (not vanishing) source region. The graphical style of the arc indicates the type of the relationship: navigational, temporal, or spatial as depicted in Fig. 9.

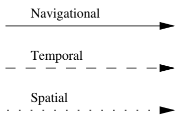


Fig. 9. Types of relationships

Navigational. The navigational relationships were introduced to represent the classical hyperlinks (so called 'click-able' links). After the user clicks on the source region the target region is invoked and the source region either vanishes or stays, depending on its type.

There is a possibility to guard the execution of the link with a condition (the link is followed only if the condition is satisfied) and the possibility to specify an event when the link should be followed. The default event for navigational links is the mouse-click event, but the designer can choose different events (e.g. mouse-over). Navigational links are graphically depicted with solid arcs.

Temporal. Temporal relationships were introduced in order to express the notion of time in the specification of multimedia presentations, while the original RMM was focused on hypermedia presentations (no notion of time). Temporal links can be used to describe both intrinsic delay (a duration of a video or audio clip [8]) and presentation time (temporal links introduced by the designer). There is a time-out event associated with every temporal relationship; after the time elapses the relationship is realized (the destination region is invoked)[3].

Although from semantic point of view the temporal relationships can be considered similar to the navigational links we suggest to make a distinction mainly because of the different nature of the two (navigational links require some user interaction while temporal links do not).

Temporal links can be organized in parallel or sequential compositions. The graphical syntax is rather intuitive; temporal links are graphically described with dashed arcs.

A sequential composition is depicted as a chain of links with one starting node (starting source region) and a chain where the destination region of one link is the source region for the next link.

Two temporal relationships are considered to be in parallel composition if they are not included in the same sequential composition. In other words what is not sequential is parallel.

Figure 10 shows an example of three sequential chains (A*, B*, C*) which are organized into a parallel composition and synchronized before they enter the region D. Note that normally the starting region from a sequence is shown at once (A1, B1) but it is also possible to specify a delay with respect to the top region (C1).

[3] The synchronization can sometimes introduce an additional delay (the time-out of one 'track' elapses but it still has to wait for the other with which it is synchronized).

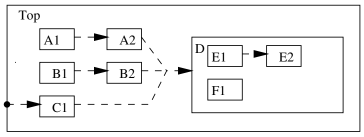


Fig. 10. Sequential and parallel composition with synchronization

Spatial. There are two reasons for introducing spatial links: the first is to provide a means of specifying the layout characteristics of a presentation. The second reason is to facilitate the description of more complex presentation elements like indices, guided tours etc. A spatial link describes a spatial relationship between two regions. The meaning of the link orientation is that the link describes a relative position of the target with respect to the source, the point of reference always being the left upper corner. The semantics of spatial links is slightly different compared to the one of navigational and temporal links. The notion of following a link is missing here, e.g. both the source and the target of a spatial link are displayed at once provided the condition and the event associated with the link permit so.

Spatial links are graphically depicted with dotted arcs.

5.3 Multi-dimensional Synchronization

Except of the synchronization of temporal relationships our framework offers also multidimensional relationship synchronization where relationships of different kinds are synchronized. This gives the designer the opportunity to specify new types of dependencies among relationships. For example in Fig. 11 a navigational link from the region 'Text' is synchronized with a temporal relationship coming from the region 'Audio'. The target region is invoked only after both events from the involved relationships (mouse-click, time-out) occur.

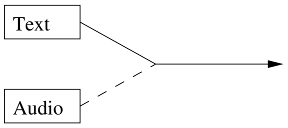


Fig. 11. Multidimensional relationship synchronization

5.4 Ordering of Relationships

A region with an entity attribute is instantiated during the presentation to several regions with concrete values of the attribute.

In order to be able to specify relationships among these instances we introduce the so-called self-relationships at the type level. Since there might be more of such relationships needed to describe the desired presentation structure, there must be a unique order in which they are applied. To facilitate this, the ordering of relationships was introduced. In Fig. 12 we provide two examples to illustrate this approach. Presentation diagrams are depicted on the left and the respective screen renderings on the right.

Figure 12 (top) describes a bulleted index region. It consists of a region with a constant attribute depicting the bullet ('·') right to which is positioned the region 'item' containing the 'N.name' attribute. From this region there is a link to the region called 'info' containing the 'N.description' attribute. The 'bullet' and the 'item' region are grouped together in the region called 'enum', which has a spatial relationship (with the lower priority 1) to itself saying that the next pair bullet-item will be placed below. In other words the constraint introduced by this relationship is taken into account after applying the constraints from the relationships that have the higher priority (0).

Figure 12 (bottom) describes a bookcase region. It consists of the 'shelf' region, which contains the 'thumb' region that has a navigational link to the 'picture' region and also a link to itself saying that the next thumb will be placed to its right. This rule is applied until there is no room left in the shelf (the horizontal dimension of the 'shelf' region). When the shelf is full we proceed with the next one by applying the self spatial relationship of the 'shelf' region which has the lowest priority (2).

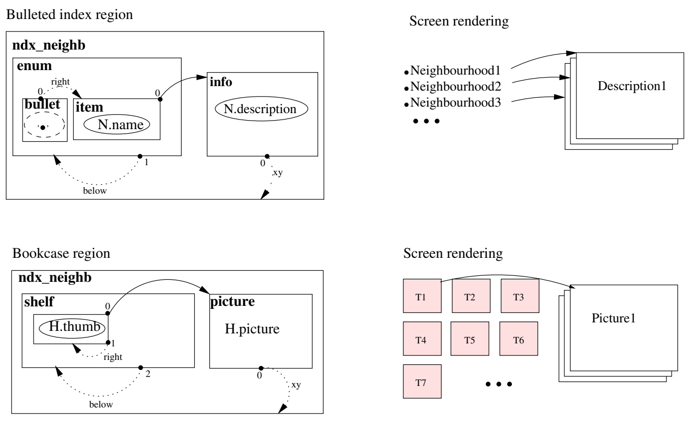


Fig. 12. Bulleted index region. Bookcase region

6 User Adaptation

In our framework we try to address the (semi-) automatic generation and design of multimedia presentations. Characteristic for the generation process is the support of adaptation.

The system can adapt to the user by considering the user's browsing behavior. By observing the navigation behavior the system can infer (based on intelligence specified by the application's author) the user's knowledge and adapt the presentation on the basis of this assumed knowledge.

There is also adaptation to the platform the user is using (platform dependence): if for example the system detects that the user is using a PC with a large screen or a small hand-held device, then the system could adapt the presentation to this platform.

Slices or slice relationships are available or accessible only after the conditions hold. While the specification of the actual adaptation process is beyond the scope of this paper (AHAM model from [10]), the concepts (in terms of AHAM) are specified in this framework as slices and slice relationships. Hence, these slices and slice relationships carry (in the conditions) part of the knowledge on which the adaptation is based. In order to specify hypermedia presentations, especially in the context of automatic design, it is necessary to describe which concepts are part of the design. Our framework allows to specify exactly that.

7 Conclusions

The research described in this paper targets the support of automated hypermedia presentation design (in the context of Web-based Information Systems). Specifically, for applications involving data that change frequently, the design of hypermedia presentations requires a structured approach. This paper discusses a methodology that guides through the process of specifying the presentation generation process. The methodology is inspired by RMM and overcomes some of RMM's shortcomings for the application to automated presentation generation. It also shows the specification techniques used in the steps of the methodology.

Our methodology distinguishes three steps in the design process. First, a logical application design is made to organize the data elements at a logical level, even independent of the platforms used for the presentation. Secondly, at the logical level some of the specific details related to the platform that is used are taken into consideration: this leads to a platform-dependent logical application diagram. The third step in the process realizes the logical decisions made in the first two steps, by carefully choosing representations for the different constructs. At this level, the dimensions of hyperlink navigation, space and time are included in the specification. This paper concentrates on the specification of the different design decisions and illustrates them using the diagrams.

In further work we will extend these specification techniques with other constructs that allow for the specification of some of the other relevant details. In the Dynamo project we are experimenting with different kinds of applications such as the Electronic TV-Program Guide. In these experiments we learn how different aspects play a role in the automated hypermedia presentation design process. We aim at including more of these aspects in this specification framework. One example relates to the use of constraints (rules) in the specification of spatial relationships. Such constraints might not (easily) lead to a solution, or even influence other design decisions (made earlier in the design process). Other aspects relate to the combination of different kinds of relationships, or the construction of 'virtual' slices or regions.

In the Dynamo project the adaptation is an important feature. The conditions associated with slices and regions allow for implementing different aspects of adaptation, such as user adaptation (navigation) and platform adaptation. The Hera prototype is developed incrementally, adding new features to the methodology enables further verifying whether this mechanism suffices in concrete applications.

References

1. Balasubramanian, P., Isakowitz, T., Stohr, E.A. : RMM: A Methodology for Structured Hypermedia Design. Communications of the ACM, 38(8):33-44, August 1995.
2. Barbosa, S.D.J., Rossi, G., Schwabe, D.: Systematic Hypermedia Application Design with OOHDM. Hypertext, 1996.
3. Bieber, M., Isakowitz, T., Vitali, F.: Web Information Systems. Communications of the ACM, 41(7):78-80, July 1998.
4. Brusilovsky, P., De Bra, P., Houben, G.J.: Adaptive Hypermedia: From Systems to Frameworks. ACM Computing Surveys, 31(4es), Article No. 12, December 1999.
5. Ceri, S., Fraternali, P., Bongio, A.: Web Modeling Language (WebML): a modeling language for designing Web sites. WWW9, 2000.
6. Fraternali, P., Paolini, P.: Model-Driven Development of Web applications: the AutoWeb system. ACM Transactions on Information Systems, 18(4):323-382, October 2000.
7. Garzotto, F., Paolini, P., Schwabe, D.: HDM - A Model-Based Approach to Hypertext Application Design. ACM Transactions on Information Systems, 11(1):1-26 , January 1993.
8. Bulterman, D.C.A., Hardman, Mullender, K.S.L., Rutledge, L., van Ossenbruggen, J.: Do You Have the Time? Composition and Linking in Time-based Hypermedia. Hypertext and Hypermedia, 1999.
9. De Bra, P., Houben, G.J.: Automatic Hypermedia Generation for ad hoc Queries on Semi-Structured Data. Digital Libraries, 2000.
10. De Bra, P., Houben, G.J., Wu, H.: AHAM: A Dexter-based Reference Model for Adaptive Hypermedia. Hypertext and Hypermedia, 1999.
11. Diaz, A., Isakowitz, T.: RMCase: A Tool To Design WWW Applications. Fourth International World Wide Web Conference, 1995.
12. Houben, G.J.: HERA: Automatically Generating Hypermedia Front-Ends for Ad Hoc Data from Heterogeneous and Legacy Information Systems. Engineering Federated Information Systems, 2000.
13. Isakowitz, T., Kamis, A., Koufaris, M.: Extending RMM: Russian Dolls and Hypertext. HICSS, 1997.

14. Isakowitz, T., Kamis, A., Koufaris, M.: Reconciling Top-Down and Bottom-Up Design Approaches in RMM. Workshop on Information Technologies and Systems, 1997.
15. Isakowitz, T., Kamis, A., Koufaris, M.: The Extended RMM Methodology for Web Publishing. New York University, 1998. Available online at `http://rmm-java.stern.nyu.edu/rmm/`.

Efficiently Mapping Integrity Constraints from Relational Database to XML Document[1]

Xiaochun Yang, Ge Yu, and Guoren Wang

Dept. of Computer Science & Engineering, Northeastern University
Shenyang, 110004, P.R.China
yangxc73@263.net {yuge,wanggr}@mail.neu.edu.cn

Abstract. XML is rapidly emerging as the dominant standard for exchanging data on the WWW. Most of application data are stored in relational databases due to its popularity and rich development experiences over it. Therefore, how to provide a proper mapping approach from relational data to XML documents becomes an important topic. Integrity constraints are useful for semantic specification that plays the important roles in relation schema definition. The existing XML schema language does not define general constraints and maintaining method for integrity constraints. So how to use XML to express and maintain integrity constraints especially the advanced integrity constraints, e.g., general constraints of relational data is one of challenge research issues. In this paper, a novel mapping approach is proposed to map relation data to XML document with active nodes, XML_A, and extended DTD with constraints, DTD_C. The ability to maintain integrity constraints makes our approach more effective than other approaches.

1 Introduction

With the explosive growth of the WWW, the requirement for sharing information becomes one of key problems. As a standard for exchanging business data on the WWW [1], XML should provide the ability of expressing data and semantics among heterogeneous data sources on the WWW. Therefore, transformation from relational data to XML data becomes the major research problem in the field of current information exchanging, sharing and integration.

Observe that the ID /IDREF(s) mechanism of DTD is too simple to express integrity constraints [2]. If the integrity semantics are implicitly expressed only by ID/IDREF(s) attribute values, the complete semantics of relational data can not be captured in the mapped XML data. However how to express integrity constraints in XML is a very important research issue because they play the key roles in specifying semantics [3], maintaining data consistency [4,5], optimizing queries [6,7], and integrating information [8-10]. Although XML Schema [11] identifies some commonly

[1] This work is partially supported by the Teaching and Research Award Program for Outstanding Young Teachers in Higher Education Institutions and the Cross Century Excellent Young Teacher Foundation of the Ministry of Education of China, the Natural Science Foundation (69803004) of China.

recurring schema constraints and incorporates them into the language specification, it does not provide a suitable method to maintain the integrity constraints. An integrity constraint is a condition specified on a database schema to restrict the data that can be stored in an instance of the database. In the relational database, SQL is used to define integrity constraints including domain constraints, referential integrity constraints, and general constraints.

A number of recent proposals aimed at transformation from popular databases to XML documents. The YAT system [10] developed by INRIA is a semistructured data conversion and integration system that adopts XML as a middleware model for wrappers, mediators communicate data, structures and operations. IBM Almaden Research Center in the literature [12] characterized the solution space based on the main differences between XML documents and relational tables, studied on the mapping problem at data level and provided a significant performance benefit. Pennsylvania University in the papers [13,14] studied constraint implication problems, concerning path and type constraints, object identifier, and inverse reference. However, as we known, the existing researches have the following problems to be studied further.

1. Literatures [12,13,14,15] only provided mapping approaches for referential integrity constraints for object-oriented and relational database. While literature [14] mainly considered the issue of data level mapping rather than schema level or higher semantic level.
2. Mapping Composite Keys and Composite Foreign Keys. The approaches [12-14] have another common problem that they provided mapping approaches only for simple keys and simple foreign keys, not for composite keys and composite foreign keys.
3. How to provide a maintenance mechanism for integrity constraints in the mapped XML document is still an problem, e.g., mapping options for enforcing referential integrity constraints. In a relational database, if a referential integrity constraint is defined, any command executing on the relational database should be evaluated first. If the evaluation is false, the command will be rejected. However, all above approaches did not consider the enforcement of referential integrity constraints in the mapped XML document.
4. Mapping General Constraints. General constraints are advanced constraints that they provide more flexibility to user to decide which operations should be executed to maintain consistency of relational data. By far, how to map general constraints is an still open problem and there is no suitable approach to solve it as the best of our knowledge.

Relational database adopts DDL, subset of SQL, to define relation schemas, while XML uses DTD and XML Schema to describe structures of documents. DTD does not provide any more semantic constraints besides unary ID and IDREF(s) attributes to represent referential mechanism, so it is not sufficient to express and maintain integrity constraints in relational database. XML Schema does not define general constraints and maintaining method for integrity constraints.

In order to provide an effective mechanism to solve the above problems, this paper considers the richness and complexity of integrity constraints, presents a Document Type Definition with Constraints(DTD_C) and corresponding XML document with Active Nodes(XML_A), and designs a DTD_C based mapping approach, named ICMAP to map relational data and its integrity constraints into XML_A document. DTD_C is an extended DTD that is used to define the syntax and integrity constraints. XML_A is an

XML document with active nodes. It conforms to DTD_C definition whose instance still conforms to XML grammar. ICMAP only extends DTD, without changing the primary XML document instance. As for composite keys/foreign keys, in ICMAP we only change the constructing method for attribute values, reduce the scope and type of an ID/IDREF attribute by specifying key attributes. Based on XML_A, we provide a mechanism to transform different options to maintain referential integrity constraints, and map advanced integrity constraints such as table constraints and assertion.

The main contributions of this paper are listed as follows. (1) ICMAP supports mapping for composite keys and composite foreign keys. (2) ICMAP provides document with active nodes to maintain integrity constraints. (3) ICMAP provides indices for multiple keys in XML data. (4) ICMAP proposes an approach to maintain integrity constraints on XML instances. (5) ICMAP presents mechanism to map general constraints. All the above characteristics make it remarkable than the other existing schema definition language [16].

The rest of the paper is organized as follows. Section 2 overviews the representation ability of relational databases and XML for integrity constraints. Section 3 proposes DTD_C and XML_A, and gives the architecture for mapping integrity constraints from relational data into XML data. Section 4 presents the ICMAP approach involving different rules to map basic table, keys, foreign keys and options for enforcing referential integrity constraints. Section 5 proposes mapping rules for general constraints and Sect. 6 gives the constraints reducing rules to optimize structure of document. Finally, Sect. 7 concludes the whole paper.

2 Overview Integrity Constraints in RDB and XML

In relational databases, entities and relationships among entities are defined as flat table, and integrity constraints among data are established implicitly through SQL. In SQL-92 specification, integrity constraints include referential integrity constraints and foreign key constraints. Key constraints and foreign key constraints are classified as referential integrity constraints. General constraints is regarded as advanced constraints that they provide more flexibility to user to decide which operations should be executed to maintain consistency of relational data. Current relational database systems support such general constraints in the form of *table constraints* and *assertion* [17]. Table constraints are associated with a single table and are checked whenever that table is modified. In contrast, assertions involve several tables and are checked whenever any of associated tables is modified.

Here is a simple example of teaching database. *Char* and *integer* distinguish the domain constraints of each attribute, *primary key* defines keys of each table, *foreign key* defines foreign keys referring to other tables. In the example, attribute *ano* is the primary key of table *advisor*, while *ano* field of table *student* is a foreign key and refers to table *advisor*. As we known, the insertions of *student* tuples that violate referential integrity will be rejected. And SQL-92 provides several alternative ways to handle the *deletion*/*update* of *advisor* tuples that violate referential integrity by choosing the four options including *no action*, *cascade*, *set default*, and *set null*.

Example 1.

```
Create table advisor (ano char(10) NOT NULL,
 aname char(20) NOT NULL, primary key(ano));

Create table student (sno char(10) NOT NULL,
 age integer, ano char(10), primary key(sno),
 foreign key(ano) references advisor,
 on delete cascade, on update no action);
```

Different from relational databases, XML adopts a hierarchical format having arbitrary nested structures with a singleton root element, and uses XML Schema as the schema definition language, provides type definition, ID/IDREF(s), keys/foreign keys to support referential mechanism. Each element has a tag associated with it. An element can have attributes and values or nested sub-elements. ID annotation indicates that the associated elements should be uniquely identified by it. IDREF(s) is/are (a) logic pointer(s) referring to the associated elements with same ID attribute value. ID and IDREF(s) are both based on unary attributes.

Although ID/IDREF(s) are similar to keys/foreign keys in relational databases, they are insufficient to express some implicit semantics. XML Schema only supports key/foreign key constraints by using *unique* tag. It can not define general constraints and maintenance mechanism so far. And XML Schema is too rigid in the definition and language specification [16].

3 XML Documents with Active Nodes

Definition 1 Active Node. An Active Node is an element node with the ability to evaluate the value of elements in an XML. Let the active node be e_A, the expression of e_A is:

```
eA = <tag attribute-dcl> value </tag>
```

The BNF statements of *attribute-dcl* are listed as follows:

```
attribute-dcl := general-attr* active-attr*
general-attr := ID-dcl* IDREF-dcl* IDREFs-dcl*
active-attr := event-dcl {condition-dcl* action-dcl}+
event-dcl := "event =" <identifier>
condition-dcl := "condition =" <identifier>
action-dcl := "action ="<identifer>{","<identifer>}+
```

The keyword *event*, *condition* and *action* are used to describe the active nodes. They constitute an ECA rule, i.e., *on event if condition then action*. The values of *event* are basic operations on an XML, such as *insert*, *delete* and *update*. The keyword *condition* records the element types that will be checked. The values of *action* are Java functions that will be dynamically loaded and interpreted at the runtime.

Definition 2 DTD with Constraints (DTD$_C$). A DTD$_C$ is an extended DTD with constraints. A DTD$_C$ is denoted by (T, r, A, K, V, Σ), where:

- T records type definitions, including in-line and user defined data types. For each type $\tau \in T$, let Ext(τ) be a set of instances labeled by τ in an XML document;
- r is the element type of the root element in the XML document, $r \in T$;
- $A(\tau)$ records all attributes of type τ;
- K is a partial function identifying attribute types in the XML document: $\bigcup_{\tau \in T} A(\tau) \rightarrow$ {ID,IDREF, IDREFs, OP}. OP represents an active node;
- V is a value constraint function $Ext(\tau) \rightarrow A(\tau)$, denotes that the values of $A(\tau)$ are originated from $Ext(\tau)$;
- Σ is a set of basic XML constraints recording all kinds of constraints that can be mapped into XML document instances.

Definition 3 ID Constraint. Suppose $l \in A(\tau)$, if $\forall x,y \in Ext(\tau)(x.l = y.l \rightarrow x = y)$, then attribute l is an ID attribute of type τ. An ID constraint can be expressed as Formula (1).

$$\tau.\mathrm{l} \rightarrow_{\mathrm{ID}} \tau \tag{1}$$

In XML documents, references are based on unary attributes, hence $l \in A(\tau)$. Formula (1) denotes that if $\tau.l \rightarrow \tau$, then for any $l' \subseteq l$, we have $\tau.l' \not\rightarrow \tau$, i.e., l represents primary key of type τ.

Definition 4 IDREF Constraint. Suppose $l \in A(\tau)$, $l' \in A(\tau')$, $\tau'.l' \rightarrow \tau'$, if $\forall x \in Ext(\tau)$ $\exists y \in Ext(\tau')(x.l = y.l')$, the attribute l of type τ is the IDREF attribute referring to type τ'. An IDREF constraint can be defined as Formula (2). Clearly, if there is an IDREF constraint $\tau.l \rightarrow_R \tau'.l'$ in Σ, constraint $\tau'.l' \rightarrow_{ID} \tau'$ must be true.

$$\tau.\mathrm{l} \rightarrow_{\mathrm{R}} \tau'.\mathrm{l}' \tag{2}$$

Definition 5 IDREFs Constraint. Suppose $l \in A(\tau)$, $l' \in A(\tau')$, $\tau'.l' \rightarrow \tau'$, if $\forall x \in Ext(\tau)$ $\exists Y \subseteq Ext(\tau')(x.l \in Y.l')$, the attribute l of type τ is the IDREFs attribute referring to type τ'. An IDREFs constraint can be defined as Formula (3). Similarly, if there is a IDREFs constraint $\tau.l \rightarrow_{RS} \tau'.l'$ in Σ, constraint $\tau'.l' \rightarrow_{ID} \tau'$ must be true.

$$\tau.\mathrm{l} \rightarrow_{\mathrm{RS}} \tau'.\mathrm{l}' \tag{3}$$

Definition 6 Key Constraint. Suppose τ, $sub\tau \in T$, $sub\tau$ is the subtype of τ, Formula (4) defines a key constraint that assert $sub\tau$ is a key of τ.

$$\tau.\mathrm{sub}\tau \rightarrow_{\mathrm{K}} \tau \tag{4}$$

Definition 7 Foreign Key Constraint. Suppose τ, $sub\tau, \tau_1, sub\tau_1 \in T$, $sub\tau$ is the subtype of τ, $sub\tau_1$ is the subtype of τ_1. If $sub\tau$ and $sub\tau_1$ have the same type, Formula (5) defines a foreign key constraint that assert $\tau.sub\tau$ is a foreign key referring to $\tau_1.sub\tau_1$.

$$\tau.\mathrm{sub}\tau \rightarrow_{\mathrm{FK}} \tau_1.\mathrm{sub}\tau_1 \tag{5}$$

The DTD$_C$ definition of Example 1 is listed as follows.

```
DTDC = T(r)={advisor*, student*}
 T(advisors)={ano,aname)
 T(students)={sno,age,ano}
 T(advisors.ano)=string
 T(advisors.aname)=string
 T(students.sno)=string
 T(students.age)=integer
 T(students.ano)=string
 A(advisor)=(no)
 A(student)=(no,ref)
 K(advisor.no)=ID
 K(student.no)=ID
 K(student.ref)=IDREF
 Σ={advisor.ano→K advisor, student.sno→K student,
 student.ano→FK advisor, student.no→ID student,
 advisor.no→ID advisor, student.ref→R advisor}
```

Definition 8 Document with Active Nodes (XML$_A$). An XML$_A$ is an *XML document* having active nodes besides data nodes. An XML$_A$ document (*E*, *Av*, *value*, *entry*) can be said as a document conforming to DTD$_C$, if and only if there is a mapping *f* : $E \cup Av \cup \{V\} \rightarrow T \cup \bigcup_{\tau \in T} A(\tau)$, such that:

- *E* is a set of instances of all elements conforming to *T*. For each regular expression $\alpha \in E$, $\alpha ::= \alpha^* \mid \alpha^+ \mid \alpha? \mid \alpha+\alpha \mid \alpha,\alpha \mid \varepsilon \mid$ atom value, ε denotes the empty value, "*" stands for the Kleene closure, "+" for at least one α, "?" for zero or one α, and "," stands for the concatenation;
- *Av* is a set of all attribute values of $A(\tau)$;
- *V* is the set of value of all atomic values, ∀value∈ *V*, *f* (value)∈ *T*;
- *Entry* is the root of the XML$_A$ document, *f* (r)=*entry*;
- XML$_A$ logically implicates the constraint set Σ.

Figure 1 illustrates an architecture of automatically mapping from relational data into XML data. Schema-Extractor and Data-Extractor are charging with extracting schema and data from relational database respectively. By using mapping rules given in the following sections, Schema-Translator maps relation schema into DTD$_C$ automatically. Based on the mapped DTD$_C$ and mapping rules, Data-Translator translates the result form Data-Extractor and produces XML$_A$ document.

The XML$_A$ is a mapped XML document that is defined to maintain data consistency required by integrity constrains. IC package is a Java based XML service library package that can be loaded into Data-Translator at runtime. The package provides core XML capabilities including a fast XML parser with optional validation and an inmemory object model graph conforming to XML$_A$. By using this way, we can filter query commands in an XML$_A$ element. Any opertions violating integrity constraints over XML$_A$ should be checked out and rejected. Java is chosen as the function description language because of Java's support for portability, its flexibility as a high level language, and its support of dynamic linking/loading, multi-threading and standard libraries. In an XML$_A$, when the basic operation described by the active

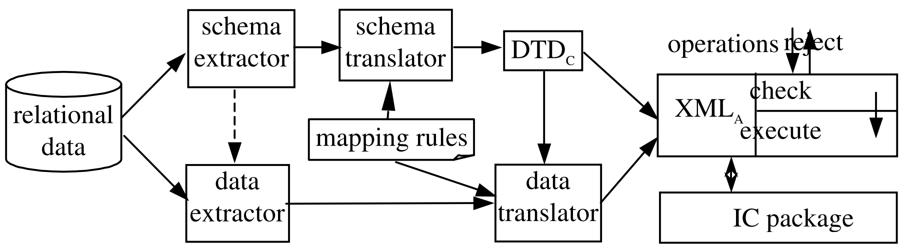


Fig. 1. Transform data from RDB to XML$_A$ with the ability to maintain data consistency

node of the attached elements is invoked, the corresponding conditions will be evaluated to decide whether the actions of checking integrity constraints are executed.

4 Referential Integrity Constraints Mapping

The mapping approach for keys and foreign keys has been fully discussed in [15]. In this paper we focus on maintaining foreign keys and mapping advanced constraints of SQL. In order to give a completely mapping rules, thebasic mapping rules are simply introduced here.

4.1 Basic Mapping

Rule 1 Basic Table Mapping. Given a table R(A) in a relational database. $\forall a_1,a_2,\ldots,a_n \in A$, the mapping rule for the basic table is listed as follows.

- The table name and attribute name are mapped into the types of the associated elements of XML$_A$: $T(R)=\{a_1, a_2,\ldots, a_n\}$;
- All tuples of table R are mapped into sub-element types of root element type: $T_{new}(r) = T_{old}(r) \cup \{R^*\}$;
- All attribute values of table R are mapped into sub-element types of type R: $T_{new}(R)=\{ a_1, a_2,\ldots, a_n \}$;
- $\Sigma_{new} = \Sigma_{old}$;
- According to the DTD$_C$, construct XML$_A$. Each tuple in table R is translated into a sub-tree with three levels. The top level of the sub-tree is an element labeled by tag R, the middle level are elements labeled by a_i, and the bottom level are atom value originated from attributed values from a R tuple. All sub-trees have the same root node *entry*.

Rule 2 Key Mapping. Given a table R(A, B) in a relational database. Let X denote the key of R. $\forall a_1,a_2,\ldots,a_n \in A$, $\forall b_1,b_2,\ldots,b_m \in B$, the mapping rule for key A is given as follows.

- Add constraints into the constraint set: $\Sigma_{new}= \Sigma_{old} \cup \{R.A \rightarrow_K R\}$;
- If A is the primary key of R
  - Let l be ID attribute of R to express primary key: $A(R)=\{l\}$, $K(R.l)=ID$;

  - The value of l is constructed by the value of A: $V(\text{R}.l)= \sum_{i=1}^{n} a_i$;
  - Add constraints into the constraint set: $\Sigma_{new}= \Sigma_{old} \cup\{\text{R}.l \rightarrow_{\text{ID}} \text{R}\}$;
- Constructing indices for keys $\{a_i\}$, the algorithm is given as follows.

```
Algorithm con_index(cur:current node,ind:index of keys)
Begin
 for each keys of cur
 node:=find(ind,cur.key);
 if(node=NULL)
 insert_value(ind,cur.key,node);
 endif
 endfor
end
```

Useful relational techniques is expected to exploited in XML_A. While processing a key, an index is built to facilitate query in XML_A. maps the values of keys to the ID attribute of the element.

In relational databases, a table with foreign keys referring to other tables can be an entity as well as a relationship between entities. For convenience, we will use three tables R(A, B), S(C, D, E) and Q(F, G) to represent multiple tables R_i, S_j and Q_k($1\leq$ i $\leq$ n,$1\leq$ j $\leq$m,$1\leq$ k $\leq$ p) respectively.

Rule 3 Foreign Key Mapping. Given tables R(A, B), S(C, D, E) and Q(F, G) in a relational database, and suppose attributes A and F are keys of tables R and Q respectively, attributes D and E are foreign keys referring to tables R and Q respectively. A, B,…, and G can represent an attribute or a set of attributes: $\text{A}=\{a_1,a_2,\ldots,a_n\}$, $\text{D}=\{d_1,d_2,\ldots,d_l\}$, $\text{E}=\{e_1,e_2,\ldots,e_n\}$, $\text{F}=\{f_1,f_2,\ldots,f_l\}$, then the mapping rule for foreign key is given as follows.

- Add foreign key constraints: $\Sigma_{new}=\Sigma_{old}\cup\{\text{S.D}\rightarrow_{\text{FK}} \text{R.A}, \text{S.E}\rightarrow_{\text{FK}}\text{Q.F}\}$;
- If A and F are primary keys of R and Q respectively
  - Let l_d, l_e be IDREF attributes of S to express foreign keys referring to R, Q in table S respectively: $A(\text{S})= A(\text{S})\cup\{l_d, l_e\}$, $K(\text{S}.l_d)$ = IDREF, $K(\text{S}.l_e)$ = IDREF;
  - Let l_R, l_Q be IDREFs attributes of R, Q respectively to express inverse reference to S: $A(\text{R})=A(\text{R}) \cup\{l_R\}$, $A(\text{Q})= A(\text{Q}) \cup\{l_Q\}$, $K(\text{R}.l_R)$ = IDREFs, $K(\text{Q}.l_Q)$ = IDREFs;
  - Construct IDREF attributes l_d, l_e in table S: $V(\text{S.l}_\text{d})=\sum_{i=1}^{l} d_i$, $V(\text{S.l}_\text{e})=\sum_{i=1}^{n} e_i$;
  - Add foreign key constraints into the constraint set: $\Sigma_{new}=\Sigma_{old}\cup\{\text{S}.l_d\rightarrow_{\text{R}} \text{R}.l', \text{S}.l_e \rightarrow_{\text{R}} \text{Q}.l'', \text{R}.l_R\rightarrow_{\text{RS}}\text{S}.l', \text{Q}.l_Q\rightarrow_{\text{RS}}\text{S}.l'\}$;
  - Constructs IDREFs attribute values from K and Σ_{new} of the mapped DTD_C, the algorithm is referred to [15].

4.2 Options Mapping for Enforcing Foreign Key Constraints

The four options described in Example 1 are specified as part of the foreign key declaration. The default option is *no action*, which means that the action (*delete* or *update*) violating constraints is to be rejected.

Rule 4 Options Mapping. Given tables R(A, B), S(C, D, E) in a relational database. Let A be the key of table R, let D be the foreign key referring to table R, let M={delete, update} be the operation set of R tuples, and let O={o_1,o_2,…,o_n} be the set of options on *M*. The mapping rule for options is given as follows.

- If *delete*∈*M*, let $l_{Re,}$ l_{Ra} be event and action attributes of R respectively, the value of R.l_{Re} be *delete*: $A(\mathrm{R})_{new} = A(\mathrm{R})_{old} \cup \{l_{Re}, l_{Ra}\}$, $K(\mathrm{R}.l_{Re}) = K(\mathrm{R}.l_{Ra})$ = OP, $V(\mathrm{R}.l_{Re})$=delete;
- If *update*∈*M*, let $l_{Ae,}$ l_{Aa} be event and action attributes of A respectively, the value of A.l_{up} be *update*: $A(\mathrm{A})_{new}=A(\mathrm{A})_{old}\cup\{l_{Ae},l_{Aa}\}$, $K(\mathrm{A}.l_{Ae})$=OP, $V(\mathrm{A}.l_{Aa})$=update;
- The values of action attribute R.l_{Ra} and A.l_{Ae} are assigned according to the different operations belongs to *M* and different options on *M*. The comparison table is listed in Table 1.

Table 1. Options and function pair

Operation	option	function	option	function
Deletion operation	cascade	delete_c	set default	delete_d
	no action	delele_n	set null	delete_u
Update operation	cascade	update_c	set default	update_d
	no action	update_n	set null	update_u

The eight functions listed in Table 1 are implementations of active nodes. The mapped XML$_A$ of Example 1 and the algorithm of *delete_c* are listed in Example 2.
Example 2.

```
<root>
 <advisor ID="001" IDREF="s001" IDREF="s002"
 event="delete" action="delete_c">
 <ano event="update" action="update_n">001</ano>
 <aname> … </aname>
 </advisor>
 <student ID="s001" IDREF="001">
 <sno> … </sno>
 </student>
 …
</root>
```

```
Algorithm delete_c(cur:current node,ind:index of keys)
Begin
 for each fkeys of cur
 refNodes := GetRefNodes(ind,cur.fkey);
 num := 0;
 for each refNodes rn of cur
 DeleteRefNodes(rn); num := num+1;
```

```
  endfor //search over
  DeleteAllNodes(cur);
  endfor
 end
```

5 General Constraints Mapping

5.1 Table Constraints Mapping

General constraints are specified over a single table using *table constraints*, which have the form *CHECK* conditional-expression. Look at Example 3:

Example 3.

```
 Create table student (sno char(10) NOT NULL,
  age integer, primary key(sno),CHECK (age >= 16);
```

Rule 5 Table Constraints Mapping. Given a table R(A, B, C) in a relational database, and suppose B={b_1,b_2,…,b_n} are defined in CHECK conditional-expression. The mapping rule for table constraints is given as follows.

- Let $l_{Re,}$ l_{Ra} be event and action attribute of element R respectively, the value of R.l_{Re} be *update*: A(R) = A(R) ∪ {l_{Re}}, K(R.l_{Re}) = OP, V(R.l_{Re})=delete;
- Let $l_{Be,}$ l_{Ba} be event and action attribute of element B respectively, the value of B.l_{Be} be *update*: A(B) = A(B) ∪ {l_{Be}}, K(B.l_{Be}) = OP, V(B.l_{Be})=update;
- Let the values of R.l_{Ra} and B.l_{Ba} be the same function name performing check operation: V(R.l_{Ra})= V(B.l_{Ba})=check.

The mapped XML_A is:

```
 <root>
  <student ID="…" event="delete" action="check">
  <sno > …</sno>
  <age event="update" action="check"> 23 </age>
  </student>
 </root>
```

In Example 3, the deletions of student tuples will not violate the *CHECK* condition-expression. However, the system can not get this semantic automatically through the *CHECK* condition expression. In order to provide a general mapping approach for table constraints, *delete* command on *student* and *update* command on *student.age* should be evaluated. If the evaluation result is false, the command will be rejected. Here, *check* is a function performing data domain validation.

5.2 Assertion Mapping

The best solution of maitaining a general constraints over several tables is to create an assertion. Rule 6 gives the mapping rules for assertion.

Rule 6 Assertion Mapping. Given tables R(A, B, C) and S(D, E, F) in a relational database, and suppose an assertion is defined over tables R and S. Let B={b_1,b_2,…,b_n} and E={e_1,e_2,…,e_m} be attributes defined in CHECK conditional-expression, the mapping rule for the assertion is given as follows.

- Let $l_{Re,}$ l_{Ra} be event and action attribute of element R respectively, the value of R.l_{Re} be *delete*: $A(\mathrm{R}) = A(\mathrm{R}) \cup \{l_{Re}\}$, $K(\mathrm{R}.l_{Re})$ = OP, $V(\mathrm{R}.l_{Re})$= delete;
- Let $l_{Se,}$ l_{Sa} be event and action attribute of element S respectively, the value of S.l_{Se} be *delete*: $A(\mathrm{S}) = A(\mathrm{S}) \cup \{l_{Se}\}$, $K(\mathrm{S}.l_{Se})$ = OP, $V(\mathrm{S}.l_{Se})$=delete;
- Let $l_{Be,}$ l_{Ba} be event and action attribute of element B respectively, the value of B.l_{Be} be *update*: $A(\mathrm{B}) = A(\mathrm{B}) \cup \{l_{Be}\}$, $K(\mathrm{B}.l_{Be})$ = OP, $V(\mathrm{B}.l_{Be})$=update;
- Let $l_{Ee,}$ l_{Ea} be event and action attribute of element E respectively, the value of E.l_{Be} be *update*: $A(\mathrm{E}) = A(\mathrm{E}) \cup \{l_{Ee}\}$, $K(\mathrm{E}.l_{Ee})$ = OP, $V(\mathrm{E}.l_{Ee})$=update;
- Let the values of R.l_{Ra}, S.l_{Sa}, B.l_{Ba} and E.l_{Ee} be the same function name performing check operation: $V(\mathrm{B}.l_{Ba})$= $V(\mathrm{E}.l_{Ea})$= check.

Fox example, the assertion definition and the mapped XML$_A$ are listed as Example4.
Example4.

```
Create ASSERTION smallDepartment
CHECK ((select count(A.ano) from advisor A)+
 (select count(S.sno) form student S)<500);

<root>
 <advisor ID="…" event="delete" action="check">
 <ano event="update" action="check"> … </ano>
 <aname> …</aname>
 </advisor>
 <student ID="…"event="delete" action="check">
 <sno event="update" action="check"> … </sno>
 <age> …</age>
 </student>
 …
</root>
```

5.3 Insertion Checking

Any insertion that violate general constraints should be rejected. In Example 1, the insertion of *student* tuple should be evaluated before commit the command.

Rule 7 Adding Insertion Checking. Given an XML$_A$ document, the insertion checking should be added if and only if any of the following three conditions hold:

1. If relation R is a subordinate relation with a foreign key constraint;
2. If a table constraint is defined on relation S;
3. If an assertion is defined over relations Q_i.

The mapping rule for insertion is given as follows.

- Let l_{ope}, l_{opc}, l_{opa} be attributes of root element r: $A(\mathrm{r}) = \{l_{ope}, l_{opc}, l_{opa}\}$, $K(\mathrm{r}.l_{ope})$ =$K(\mathrm{r}.l_{opc})$ =$K(\mathrm{r}.l_{opa})$ =OP;
- Let the value of event attribute r.l_{ope} be *insert*: $V(\mathrm{r}.l_{ope})$=insert;

- Let the values of condition attribute r.l_{opc} be R, S, Q_i for the above three conditions respectively: (1) V(r.l_{opc})=R, (2) V(r.l_{opc})=S, (3) V(r.l_{opc})={Q_i};
- Let the values of action attribute r.l_{opa} be function names: (1) V(r.l_{opa})=insert, (2) V(r.l_{opa})=insert_check, (3) V(r.l_{opa_i})= insert_assertion;

For example, the root element in Example 4 is changed follows:

```
<root event="insert" condition="advisor","student"
 action="assertion_check"> …
</root>
```

6 Structure Optimization

Rules 1~7 propose general mapping rules based on integrity constraints. Intuitively, the hierarchy is more suitable for XML specification as well as the relationships in real world. Thus, some constraints can be replaced by hierarchy. By using constraints reducing rule, the size of constraint set Σ can be reduced, and the query ability can be optimaized further.

Rule 8 Constraints Reducing. Given element types R, S in an XML_A document. Assume that S is the sub-element of the root element r, e_R is a sub-elements of R, l_R is an IDREFs attribute of R, e' is the primary key of elements S, and l' is an ID attribute of S. If {R.$l_R \rightarrow_{RS}$ S.l', R.$e_R \rightarrow_{FK}$ S.e}∈Σ , let attributes l_{Ra}, l_{Sa} are action attributes of R, S respectively. The constraints reducing rule for elements R is given as follows.

- $T_{new} = T_{old}$;
- If V(R.l_{Ra})=*delete_u* or V(R.l_{Sa})= *update_u*, do not change XML_A
- Else
  Let S be sub-element of R, remove the attribute l_R and the edge between the root type r and S: T(r)= T(r)−{S*}, T(R)= T(R) ∪{S*}, A(R)= A(R)−{l_R};
- Change the values of action attribute l_{Ra} and l_{Sa} to corresponding functions name listed in Table 2;
- Reduce constraints in Σ: $\Sigma_{new} = \Sigma_{old}$−{ R.$l_R \rightarrow_{RS}$ S.l', R.$e_R \rightarrow_{FK}$ S.e }.

Table 2. Function pair

functions based on constraints	functions based on hierarchy	functions based on constraints	functions based on hierarchy
delete_c	deletec	update_c	updatec
delele_n	deleten	update_n	updaten
delete_d	deleted	update_d	updated
delete_u	—	update_u	—

The hierarchical format in XML implies that element S can not be the sub-element of elements R and Q simultaneity. When S is associated with R and Q simultaneity, we only change one pair of constraints, e.g., constraints between R and S into hierarchy, and keep the other pairs of constraints. In fact, the hierarchy like R and S can solve most of the instances in the real world, e.g., the relationships among university, department, faculty, student can be substituted for hierarchy. With multiple IDREFs in an XML_A, the order in which they are pruned may affect the final structure. We adopts a depth first ordering approach in loading and processing elements within an XML_A. The optimization mapping of Example 2 is listed as follows.

```
<root event="insert"condition="student"action="…">
 <advisor ID="001"event="delete"action="deletec">
 <ano event="update" action="update_n">001</ano>
 <aname> … </aname>
 <student ID="s001"IDREF="001"> <sno> … </sno>
 </student>
 …
 </advisor>
</root>
```

7 Conclusion

This paper proposed an XML document with active nodes, XML_A, and extended DTD, DTD_C, to solve the integrity constraints mapping problem from relational databases to XML document. More specifically, the characteristics of ICMAP are listed as follows. (1) In XML_A, the basic mappings of referential integrity constraints are preserved. (2) In order to provide an effective and convenience approach to maintain integrity constraints, the active nodes is defined to check commands on XML instance. (3) In order to enforce integrity constraints, four options mapping over basic command *delete*, *update* and *insert* were presented. (4) The times of searching nodes within a document can be reduced by building indices based on keys. (5) The mapping rules for general constraints such as table constraints and assertion were proposed. (6) The hierarchy based constructing method was adopted to express the flat relational data in XML_A that provided a good foundation for structure optimization.

The study of mapping integrity constraint provides stable foundation for information integration and query optimization. General constraints provide rich definitions for semantic constraints. So far, how to map trigger is still an open problem that is also our future work. We believe that ICMAP approach proposed here is promising and provides a good idea for our further research. The distinguished property of ICMAP is that the active node can be defined on instance level as well as schema level. The property are also useful for integrating information by customize the Java function of active nodes.

Reference

1. Bray, T., Paoli, J., Sperberg-McQueen, C.M., and Maler, E.: Extensible Markup Language (XML) 1.0 (Second Edition). W3C recommendation REC-xml-20001006 (2001) http://www.w3.org/TR/2000/REC-xml-20001006
2. Biron, P.V. and Malhotra, A.: XML schema part 1: Structure. W3C Working Draft (1999) http://www.w3.org/TR/xmlschema-2/
3. Abiteboul, S. and Vianu, V.: Rugular Path Queryies with Constraints. In Proc. 16th ACM Symp. on Principles of Database Systems (1997) 51-61

4. Labrinidis, A. and Roussopoulos, N.: WebView Materialization. In Proc. of the ACM SIGMOD Intl. Conf. on Management of Data. Dallas, Texas (2000)14-19
5. Challenger, J., Iyengar, A., and Dantzig, P.: A Scalable System for Consistently Caching dynamic Web Data. In Proc. of INFOCOM'99 (1999)
6. Calvanese, D., Giacomo, G.D., and Lenzerini, M.: What is View-Based Query Rewriting? In Proc. of the 7th Intl. Workshop on Knowledge Representation Meets Databases (2000) 17-27
7. Papakonstantinou, Y. and Vassalos, V.: Query Rewriting for Semistructured Data. In Proc. of ACM SIGMOD Intl. Conf. on Management of Data (1999) 455–466
8. Cluet, S., Delobel, C., Siméon, J., and Smaga, K.: Your Mediators Need Data Conversion! Proc. of ACM SIGMOD Conf. on Management of Data. Seattle, U.S.A. SIGMOD Record, Vol.27(2). Association for Computing Machinery, Inc., New York (1998)177-188
9. Ullman, J.D.: Information Integration Using Logical Views. In Proc. of the 6th Intl. Conf. on Database Theory. Lecture Nodes in Computer Science, Vol. 1186. Springer-Verlag, Berlin Heidelberg New York (1997) 19–40
10. Christophides, V., Cluet, S., and Siméon, J.: On Wrapping Query Languages and Efficient XML Integration. Proc. of ACM SIGMOD Conf. on Management of Data. Dallas, U.S.A. In: W.Chen, J.Naughton and P.Bernstein(eds.): SIGMOD Record, Vol. 29(2). Association for Computing Machinery, Inc., New York (2000) 141-152
11. Thompson, H.S., Beech, D., Maloney, M., Mendelsohn N.(ed.): XML Schema Part 1: Structures. W3C (2000) http://www.w3.org/TR/xmlschema-1
12. Carey, M., Lindsay, B., Pirahesh, H., and Reinwald, B.: Efficiently Publishing Relational Data as XML Documents. Proc. of 26th VLDB. Morgan Kaufmann, Cairo, Egypt (2000) 65-76
13. Buneman, P., Fan, W., Weinstein, S.: Interaction between Path and Type Constraints. Proc. of ACM Symposium on Principles of Database Systems. ACM Press. Philadelphia, U.S.A (1999) 56-67
14. Fan, W. and Siméon, J.: Integrity Constraints for XML. Proc. of ACM Symposium on Principles of Database Systems. ACM Press. Dallas, Texas (2000)
15. Yang, X. and Wang, G.: Mapping Referential Integrity Constraints from Relational Databases to XML. Proc. of 2nd Intl. Web-Age Information Management. In: Lu, H. and Yu, G. (eds.) Lecture Nodes of Computer Science, Springer-Verlag, Xi'an, China (2001)
16. Lee, D. and Chun, W. Comparative Analysis of Six XML Schema Languages. SIGMOD Record. Vol.29(3). Association for Computing Machinery, Inc., New York (2000) 76-87
17. Ramarkrishnan, R and Gehrke, H.: Database Management Systems(Second Edition). McGraw-Hill. New York, 1999.

A Web-Based System for Handling Multidimensional Information through MXML[*]

Manolis Gergatsoulis[1], Yannis Stavrakas[1,2], Dimitris Karteris[1], Athina Mouzaki[1], and Dimitris Sterpis[1]

[1] Institute of Informatics & Telecommunications
National Centre for Scientific Research (N.C.S.R.) 'Demokritos'
153 10 Aghia Paraskevi Attikis, Greece
[2] Knowledge & Database Systems Laboratory
National Technical University of Athens (NTUA), 157 73, Athens, Greece
{manolis,ystavr,mouzaki}@iit.demokritos.gr
dkart@tee.gr

Abstract. In this paper we address an issue common in the frame of WWW, namely information entities that present different facets under different *contexts* (or *worlds*). Handling such multifacet or *multidimensional* entities requires a multidimensional paradigm for Web data, which consists of representation, manipulation and presentation issues. For representing multidimensional data we employ Multidimensional XML, a markup language that incorporates *dimensions* in XML. We discuss the presentation of multidimensional data through multidimensional XSL stylesheets. We describe the design of a system that implements the basic functionality of the multidimensional paradigm, and demonstrates how a user can interact with a multidimensional document and view different variants of the document under different worlds.

Keywords: Multidimensional XML/XSL, Web Databases, Multidimensional Languages

1 Introduction

The wide acceptance of WWW was due, to a great extend, to the simplicity of its protocol (HTTP) and language (HTML). However, the need for more sophisticated functionality led to secure protocols (HTTPS), flexible languages (XML) [5,7] and ambitious visions for the future of the Web (Semantic Web [4]).

Viewing the Web as a large database, a number of query languages and data models, such as semistructured data [1,15], have been proposed. However, those models fall short when it comes to representing *multidimensional information*; that is, information that presents different facets under different contexts. Actually, there are many cases where variants of the same entity do exist. As a

[*] This work has been partially supported by the Greek General Secretariat of Research and Technology under the project "Executable Intensional Languages and Intelligent Applications in Multimedia, Hypermedia and Virtual Reality" of ΠΕΝΕΔ'99.

simple example imagine a report that needs to be represented at various degrees of detail and in various languages. A solution would be to create a different document for every possible combination. Such an approach is certainly not practical, since it involves excessive duplication of information. What is more, the different variants are not associated as being parts of the same entity. The problem of varying entities is very common in the frame of the WWW, where information providers cannot assume too much about the background *context* of the information consumers. For those reasons, models and languages suitable to represent and exchange multidimensional data over the Web are needed.

Ideas on how this problem can be tackled are given in [14,13], where a formalism called Multidimensional XML (MXML) is presented. MXML was influenced by *Intensional HTML* (IHTML) [17,6,16]. IHTML is a Web authoring language, based on and extending ideas proposed for a software versioning system in [12], that allows a single Web page to have different variants and to dynamically adapt itself to a given context. The main difference between IHTML and MXML is a projection of the difference between HTML and XML, that is, focusing on encoding structure rather than on presentation.

In this paper, we propose a multidimensional paradigm for representing and viewing context-dependent Web data. We present a comprehensive example that demonstrates the syntax and relationships of the various multidimensional components in the paradigm. We describe the architecture and design of a system that implements the basic functionality of the proposed paradigm, and discuss some implementation issues. Finally, we conclude the paper with directions for future work.

2 A Multidimensional Paradigm

The problem of Web information entities that may assume different facets under different *contexts* (or *worlds*) leads to a new paradigm for representing and viewing such entities. We refer to the new paradigm as *multidimensional paradigm*, and in this section we examine its various aspects.

2.1 The Multidimensional Approach

A widely adopted way to encode, represent, and exchange information in the frame of WWW is through eXtensible Markup Language (XML in short) [5,7]. XML is flexible enough to adapt to different domains, and capable of handling the irregularities often exhibited by Web data. XML does not address the issue of how to present information; a solution to this is offered by eXtensible Stylesheet Language (XSL in short) [2,8].

XSL can be seen as a document containing instructions on how to present information in XML documents. It is important to note that XSL stylesheets are also XML documents. An XSL stylesheet can be applied to a specific XML document, and the result can be displayed by a Web browser. Actually, a number of XSL documents can be applied to the same XML document, resulting in different ways to view that XML document, or parts of it.

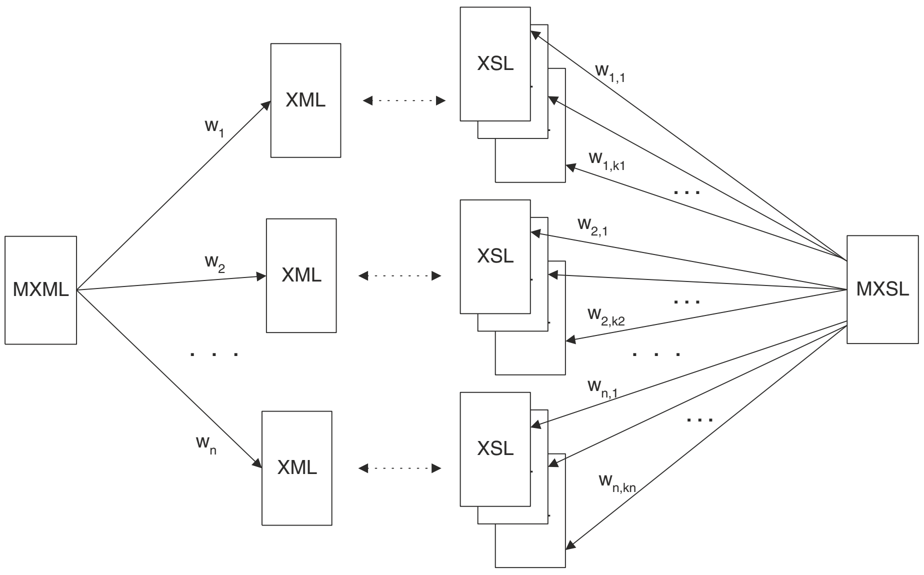


Fig. 1. Multidimensional XML, multidimensional XSL, and possible worlds

A multidimensional approach would allow a single document to have a number of variants, each holding under a specific *world*. A world is defined by giving values to a number of parameters called *dimensions*. We assume that information in such a document is encoded in a suitable markup language called *multidimensional XML* (*MXML* in short). Once a world is specified, such an MXML document can be reduced to a conventional XML document that constitutes the holding facet under that world. This world-associated decomposition of an MXML document to a number of conventional XML variants is depicted in Fig. 1, where the possible worlds for the MXML document are denoted as $w_1, w_2 \ldots w_n$.

Since XSL stylesheets are XML documents, the same principles hold for XSL stylesheets as well. A multidimensional XSL stylesheet (*MXSL* in short) encodes a set of conventional XSL stylesheet, each being the facet of the MXSL under a specific world. Like conventional XSLs, an MXSL must be associated with an XML or an MXML document. For each possible world, the holding XSL is applied to the holding XML to give the view of the document under that world. The relation between MXML and MXSL is given pictorially in Fig. 1. Note that the possible worlds for an MXSL may not be identical with those of the corresponding MXML. A number of dimensions in an MXSL may relate to the definition of alternative presentations for the same XML document. Besides, some dimensions of an MXML may also be used in the corresponding MXSL to establish a correspondence between the holding variants of MXML and MXSL.

2.2 Multidimensional XML

For supporting the multidimensional approach described above, we propose *Multidimensional XML* (MXML for short) [14,13,10], a markup language that incorporates in an elegant way multidimensional capabilities in XML. The notion of *world* is fundamental in MXML. A world represents an environment under which data in a multidimensional document obtain a substance. A world is determined by assigning values to a set $\mathcal{S}$ of *dimensions*. For each dimension d there exists a domain $\mathcal{D}_d$ over which d ranges. A **world** W is defined to be a set which, for each dimension $d \in \mathcal{S}$, contains a pair (d, u), where $u \in \mathcal{D}_d$.

In an MXML document, dimensions may be applied to elements and attributes. An element whose content depends on one or more dimensions is called *multidimensional element*, while an attribute whose value depends on one or more dimensions is called *multidimensional attribute*.

An MXML document uses *context specifiers* which are syntactic constructs that specify sets of worlds. Context specifiers qualify the variants of multidimensional elements or attributes, relating each variant to the set of worlds under which the variant becomes the holding one for the corresponding multidimensional entity.

In MXML syntax, a multidimensional element has the form:

```
<@element_name  attribute_specification>
   [context_specifier_1]
      <element_name attribute_specification_1>
          element_content_1
      </element_name>
   [/]
       . . .
   [context_specifier_N]
      <element_name attribute_specification_N>
          element_content_N
      </element_name>
   [/]
</@element_name>
```

A multidimensional element is denoted by preceding the element name with the special symbol "`@`". A multidimensional element encloses one or more *context elements* that constitute facets of that multidimensional element, holding under specific worlds specified by the corresponding context specifier. Context elements have the same form as conventional XML elements. All context elements belonging to a multidimensional element have the same name which is the name of the multidimensional element. In MXML, context elements of the same multidimensional element may have different value or structure.

To declare a multidimensional attribute we use the following syntax:

```
attribute_name = [context_specifier_1] attribute_value_1 [/]
                                     . . .
                 [context_specifier_n] attribute_value_n [/]
```

A multidimensional attribute is expressed as an attribute whose value is a set of context - value pairs. Each one of those context-associated values becomes the holding value of the attribute under the corresponding context. Therefore, under different worlds, an attribute may evaluate to different values depending on its context specifiers.

A *context specifier* is of the form:

```
dimension_1_specifier, ..., dimension_m_specifier
```

where `dimension_i_specifier`, for `i` = 1 to `m` is a *dimension specifier* of the form:

```
dimension_name  specifier_operator  dimension_value_expression
```

A *specifier_operator* is one of =, ! =, `in`, `not in`. If the *specifier_operator* is either = or ! =, the *dimension_value_expression* consists of a single dimension value. Otherwise, if the *specifier_operator* is either `in` or `not in`, the *dimension value expression* is a set of values of the form $\{\mathtt{value_1}, \ldots, \mathtt{value_k}\}$, with $k \geq 1$.

In both multidimensional attributes and elements, a context specifier may also be the reserved word "`default`". The `default` context specifier represents all worlds not covered by other context specifiers of the same multidimensional entity. The empty context specifier [ ] is called *universal context specifier*, and represents the set of all possible worlds.

A multidimensional DTD (MDTD in short) has been proposed in [10] for defining constraints on the structure of MXML documents. An MDTD allows to specify dimensions and their respective domains. Moreover, it is possible to impose different constraints under different contexts to an element structure. In this way, an MDTD can specify that an element has different structure under different sets of worlds. A graph data model for MXML has also been proposed in [10] that takes into acount contexts when representing MXML data.

2.3 Reducing MXML to XML

An important point concerning the context specifiers of a multidimensional entity is that they must be mutually exclusive, in other words, they must specify disjoint sets of worlds. This property of multidimensional entities makes it possible, given a specific world, to safely reduce an MXML document to an XML document holding under that world.

Informally, the reduction of an MXML document D to an XML document D_w holding under the world w proceeds as follows:

Beginning from the document root to the leaf elements, each multidimensional element E is replaced by its context element E_w, which is the holding facet of E under the world w. If there is no such context element, then E along with its subelements is removed entirely.

A multidimensional attribute A is transformed into a conventional attribute A_w whose name is the same as A and whose value is the holding one under w. If no such value exists then the attribute is removed entirely.

Notice that the above process can be generalized for producing a facet that holds under a set of more than one worlds. In the general case, that facet is an MXML document which is a pruned version of the original MXML document.

2.4 A Comprehensive Example

As an example of the multidimensional paradigm, consider information about a book which exists in two different editions, an English and a Greek one. In Example 1, the element `book` has six subelements. The `isbn` and `publisher` are multidimensional elements and depend on the dimension `edition`. The elements `title` and `authors` remain the same under every possible world. The element `price` is a multidimensional element whose value depends on the dimensions `edition` and `customer_type`. Note that the element `translator` has substance only under the worlds where `edition` has the value `greek`.

Example 1. Multidimensional Information about a book encoded in MXML.

```
<book>
  <@isbn>
     [edition = greek] <isbn>0-13-110370-9</isbn> [/]
     [edition = english] <isbn>0-13-110362-8</isbn> [/]
  </@isbn>
  <title>The C programming language</title>
  <authors>
     <author>Brian W. Kernighan</author>
     <author>Dennis M. Ritchie</author>
  </authors>
  <@publisher>
     [edition = english] <publisher>Prentice Hall</publisher> [/]
     [edition = greek] <publisher>Klidarithmos</publisher> [/]
  </@publisher>
  <@translator>
     [edition = greek] <translator>Thomas Moraitis</translator> [/]
  </@translator>
  <@price>
     [edition=english,customer_type=individual]<price>13.000</price>[/]
     [edition=english,customer_type=library]<price>10.000</price>[/]
     [edition=english,customer_type=student]<price>11.700</price>[/]
     [edition=greek,customer_type=individual]<price>5.000</price>[/]
     [edition=greek,customer_type=library]<price>3.000</price>[/]
     [edition=greek,customer_type=student]<price>4.500</price>[/]
  </@price>
</book>
```

The MXML in Example 2 is an MXSL stylesheet for the MXML document in Example 1. This MXSL specifies how the various facets of the corresponding MXML in Example 1 are to be presented.

The `ISBN` is shown only if the request has been made by a library, while `title` and `authors` are shown if the potential client is an individual or a student. Note how the `wrapper` element, which is defined in XSL, can also be used elegantly in MXSL to exclude the `translator` in case the request concerns the original language edition of the book. Finally, `publisher` and `price` are displayed in any case.

Notice that an MXSL may contain multidimensional versions of the elements and attributes defined within the frame of conventional XSL. There is no constraint on which dimensions participate in the MXSL context specifiers; they may also occur in the corresponding MXML document, as is the case of dimensions `customer_type` and `edition` in Example 2, or they can be different, as is the case of dimension `size` in the same example.

Example 2. An MXSL multidimensional stylesheet for Example 1.

```
<xsl:template match="/">
  <DIV STYLE=[size=large]"font-size:22pt"[/][size=normal]
    "font-size:18pt"[/]>Book
  </DIV>
  <@SPAN>
    [customer_type = library]
      <SPAN STYLE="font-size:15pt">
        ISBN: <xsl:value-of select = "book/isbn"/>,
      </SPAN>
    [/]
    [customer_type in {individual, student}]
      <SPAN STYLE="font-size:15pt">
        Title: <xsl:value-of select="book/title"/>,
        Authors: <xsl:value-of select="book/authors"/>,
        <@wrapper>
          [edition=greek]
            <wrapper>
              Translator: <xsl:value-of select="book/translator"/>,
            </wrapper>
          [/]
        </@wrapper>
      </SPAN>
    [/]
  </@SPAN>
  <SPAN STYLE="font-size:15pt">
    Publisher: <xsl:value-of select="book/publisher"/>,
  </SPAN>
  <SPAN STYLE="font-size:15pt">
    Price: <xsl:value-of select="book/price"/>
  </SPAN>
</xsl:template>
```

For the world `w = {(edition, greek), (customer_type, student)}`, the MXML document in Example 1 is reduced to the conventional XML document in Example 3.

Example 3.

```
<book>
  <isbn>0-13-110370-9</isbn>
  <title>The C programming language</title>
  <authors>
    <author>Brian W. Kernighan</author>
    <author>Dennis M. Ritchie</author>
  </authors>
  <publisher>Klidarithmos</publisher>
  <translator>Thomas Moraitis</translator>
  <price>4.500</price>
</book>
```

For the world w' = {`(edition, greek)`, `(customer_type, student)`, `(size, large)`}, the MXSL stylesheet in Example 2 is reduced to the XSL stylesheet in Example 4.

Example 4.

```
  <xsl:template match="/">
    <DIV STYLE="font-size:22pt"> Book </DIV>
    <SPAN STYLE="font-size:15pt">
       Title: <xsl:value-of select="book/title"/>,
       Authors: <xsl:value-of select="book/authors"/>,
       <wrapper>
         Translator: <xsl:value-of select="book/translator"/>,
       </wrapper>
    </SPAN>
    <SPAN STYLE="font-size:15pt">
      Publisher: <xsl:value-of select="book/publisher"/>,
    </SPAN>
    <SPAN STYLE="font-size:15pt">
      Price: <xsl:value-of select="book/price"/>
    </SPAN>
  </xsl:template>
```

The result of applying the XSL stylesheet of Example 4 to the XML document of Example 3 looks like the following:

Book
Title: The C programming language, Authors: Brian W. Kernighan, Dennis M. Ritchie, Translator: Thomas Moraitis, Publisher: Klidarithmos, Price: 4.500

Finally, the analogous result for the world w' = {`(edition, english)`, `(customer_type, library)`, `(size, normal)`} would look like this:

Book
ISBN: 0-13-110362-8, Publisher: Prentice Hall, Price: 10.000

3 System Architecture

In this section we describe a prototype system that demonstrates the basic principles of the multidimensional paradigm we introduced in the previous sections. The implemented system is called MXML Web Server and is a Web server capable of handling multidimensional data encoded in MXML/MXSL.

In a typical scenario, the user requests an MXML document through a conventional Web browser, and is prompted to select values for each of the dimensions associated with a requested document. After a world has been specified by the user, the server sends the corresponding XML/XSL facet to be displayed by the browser. The user can change the values of dimensions and observe how different worlds are associated to different variants of the same multidimensional document.

An important point is that, in the multidimensional paradigm, the management of dimensions and the reduction of MXML/MXSL can take place at the server, the client, or both. Our system implements this functionality at the server side mainly for reasons of compatibility with existing Web browsers. In the general case, however, some of the dimensions can be considered at server side, to eliminate irrelevant data and reduce the size of the response, while the rest of the dimensions could be handled at client side, for reasons of privacy or for minimizing the number of subsequent requests. This flexibility could be useful and further explored in the frame of applying MXML to domains such as electronic commerce and user modeling.

3.1 Extending URLs

URL (Uniform Resource Locator) [3] is the standard way to specify a resource available on the Internet. We extend the syntax of URL in order to enable it to handle multidimensional information. The extended URL has the form:

```
http://<host>:<port>/<path><context>?<search>
```

The only difference from conventional URL is that the token <`context`> has been added, in order to incorporate dimensions. In the following example we show an extended URL that represents a request for the variant of the document named `books.mxml`, where `edition` is `english` and `customer_type` is `student`.

```
http://myserver/books.mxml[edition=english,customer_type=student]
```

3.2 Design and Operation

The system comprises the software modules illustrated in Fig. 2. These modules are: Request Analyzer, MXML Request Decomposer, View Extractor, MXML Response Composer, and Conventional Web Server. In the following paragraphs we shall briefly describe the functionality of each of those modules, as well as their submodules, and discuss their role in the system operation.

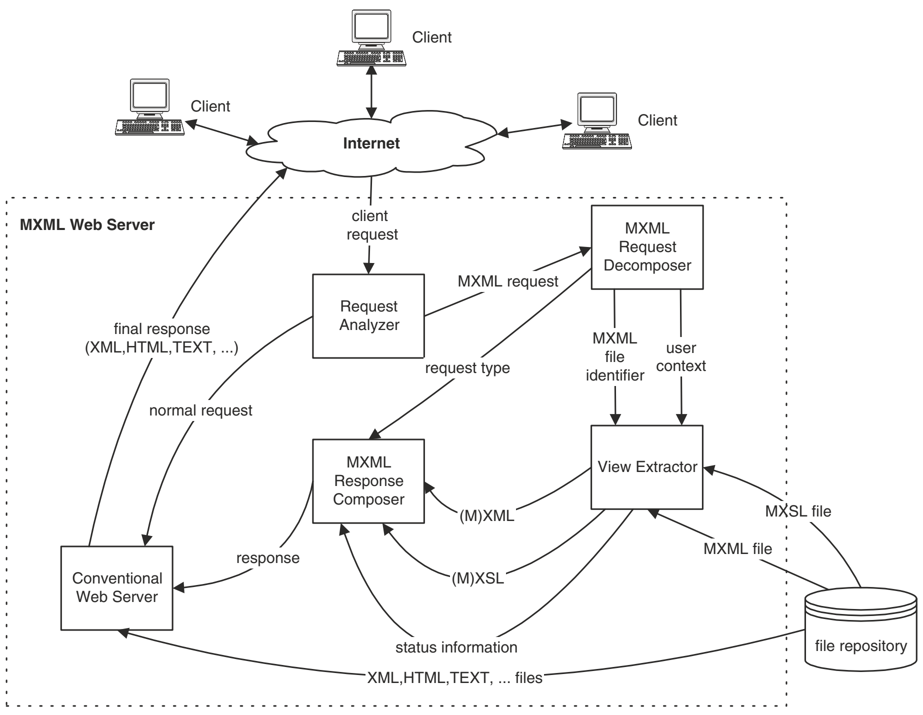


Fig. 2. MXML Web Server architecture

Request Analyzer: The Request Analyzer is responsible for determining the type of client requests. There are two types of requests; *normal* requests, which refer to HTML, XML, text files etc., and *MXML-requests*, which refer to MXML files. If the request is of normal type, then the Conventional Web Server handles it. Otherwise, if the request involves an MXML document, then the MXML Request Decomposer is invoked to serve the request.

MXML Request Decomposer: This module decomposes the MXML request into sections. The three sections that comprise the MXML request are: the MXML File Identifier, the User Context and the Request Type. The first two sections are sent to the View Extractor module, while the third is sent directly to the MXML Response Composer.

View Extractor: This module combines the information provided by the MXML Request Decomposer with the corresponding MXML or MXSL files in order to serve the request. The View Extractor consists of two sub-modules called MXML Parser and MXML Specializer, as it is depicted in Fig. 3(a).

– **MXML Parser:** The MXML Parser uses the requested MXML File Identifier, which is provided by the MXML Request Decomposer, to access the

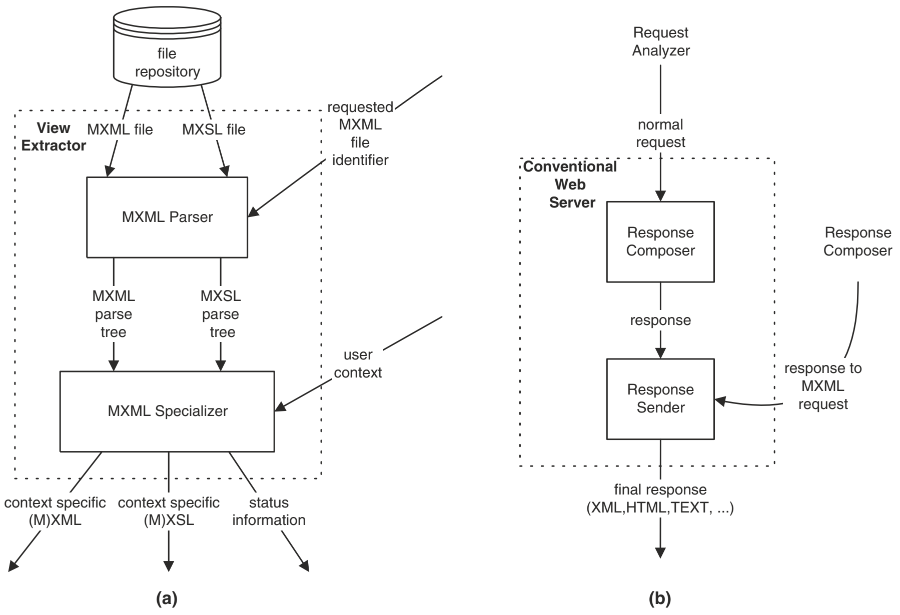


Fig. 3. (a) View Extractor's sub-modules, (b) Conventional Web Server's sub-modules

corresponding MXML or MXSL file from the File Repository. Its role is to parse the file and generate the corresponding MXML or MXSL parse tree.

– **MXML Specializer:** The MXML Specializer uses the parse tree that is generated by the MXML Parser, and the User Context, which is provided by the MXML Request Decomposer, to generate a context specific (M)XML or (M)XSL file. This is done by reducing the original MXML/MXSL file to an (M)XML/(M)XSL file that holds under the specified context. The type of the produced file is passed to MXML Response Composer through the status information.

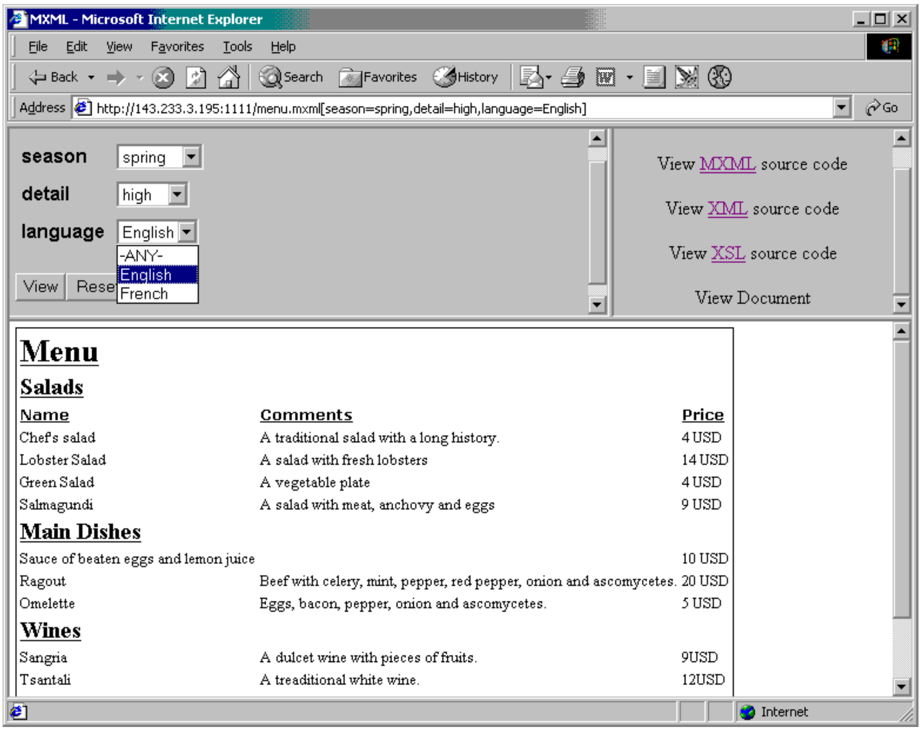
MXML Response Composer: The MXML Response Composer constructs the response and sends it to the Response Sender, which is responsible for dispatching the response to the client.

Conventional Web Server: The Conventional Web Server implements some of the essential features of a conventional Web server. It consists of two submodules, shown in Fig. 3(b). In case the type of request is *normal*, the response is constructed by the Response Composer submodule that is part of the Conventional Web Server, whereas in case the type of request is *MXML-request* the response is produced in MXML Response Composer module and is passed to the Response Sender submodule of the Conventional Web Server.

– **Response Composer:** The Response Composer analyzes the normal request, and constructs the response that is sent to the client. As it is depicted in Fig. 3(b), this module is engaged in a client request service procedure only if this request is of type *normal*.
– **Response Sender:** The Response Sender is responsible for sending the response to the client. As stated above, the response originates either from the Response Composer, or from the MXML Response Composer.

4 Implementation

The system components described in the previous section are in fact implemented as two separate programs. The MXML Web Server (except View Extractor) was developed in Java in order to take advantage of the powerful features that this language offers for network programming. The View Extractor has been implemented in ANSI C.


MXML - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites History
Address http://143.233.3.195:1111/menu.mxml[season=spring,detail=high,language=English]
Go
View Rese
-ANY-
English
French
View MXML source code
View XML source code
View XSL source code
View Document
Internet

season: spring
detail: high
language: English

Menu

Salads

Name	Comments	Price
Chef's salad	A traditional salad with a long history.	4 USD
Lobster Salad	A salad with fresh lobsters	14 USD
Green Salad	A vegetable plate	4 USD
Salmagundi	A salad with meat, anchovy and eggs	9 USD
Main Dishes		
Sauce of beaten eggs and lemon juice		10 USD
Ragout	Beef with celery, mint, pepper, red pepper, onion and ascomycetes.	20 USD
Omelette	Eggs, bacon, pepper, onion and ascomycetes.	5 USD
Wines		
Sangria	A dulcet wine with pieces of fruits.	9USD
Tsantali	A treaditional white wine.	12USD

Fig. 4. A screenshot showing a reply from MXML Web Server

Figure 4 is a screenshot of the reply of MXML Web Server to the request:

```
http://143.233.3.195:1111/menu.mxml[season=spring,
      detail=high,language=English].
```

The context specifier in this extended URL is composed dynamically by JavaScript code, based on the values selected by the user for the dimension drop-down lists, in the upper left *dimension frame*. Note that for every dimension there exists a value “ANY” in the corresponding drop-down list. The meaning of “ANY” is that no value is specified for that dimension. Consequently, if a context specifier contains one or more “ANY”, it specifies more than one worlds. If it does not contain “ANY” at all, it specifies exactly one world.

The upper-right *action frame* allows the user to view the MXML source code, and the MXSL source code. If the selected context specifies exactly one world (does not contain “ANY”) then the user may also see the XML and XSL sources of the corresponding facets under that world, plus the final output document. The source code and the final document are displayed in the lower *output frame*, which in the case of Fig. 4 shows a variant of a multidimensional menu of a restaurant corresponding to the world `w = {(season, spring), (detail, high), (language, English)}`.

The implemented system can be reached at the URL:

`http://www.iit.demokritos.gr/~mxml`

5 Conclusions and Future Work

In this paper we proposed a paradigm for handling multidimensional data in the frame of the Web. We proposed the use of MXML to represent multidimensional information, and showed how multidimensional stylesheets can be expressed in MXSL. Through a comprehensive example we demonstrated the reduction of MXML/MXSL to conventional XML/XSL under a specific world. Finally, we presented a system that implements the basic functionality of the proposed paradigm, and discussed some implementation issues.

Our future plans include the investigation on possible applications of MXML in diverse fields, such as electronic commerce and digital libraries. We also consider the representation of time dependent data through MXML [11], the representation of cartographic and GIS information where a possible dimension is *scale*, and applications where user profiling information plays an important role for delivering the right data.

Since MXML is primarily considered as a data model and data exchange format, an interesting direction is how to efficiently store and retrieve MXML data. Approaches used for XML, that employs RDBMS for storing XML fragments [9], seem promising for adaptation to MXML.

References

1. S. Abiteboul, P. Buneman, and D. Suciu. *Data on the Web: From Relations to Semistructured Data and XML*. Morgan Kaufmann Publishers, 2000.
2. S. Adler, A. Berglund, J. Caruso, S. Deach, P. Grosso, E. Gutentag, A. Milowski, S. Parnell, J. Richman, and S. Zilles. Extensible Stylesheet Language (XSL) Version 1.0. `http://www.w3.org/TR/xsl`, 2000.
3. T. Berners-Lee, L. Masinter, and M. McCahill. Uniform Resource Locators (URL). `http://www.ietf.org/rfc/rfc1738.txt`, 1994.
4. Tim Berners-Lee. Semantic web road map. `http://www.w3.org/DesignIssues/Semantic.html`, 1998.
5. T. Bray, J. Paoli, and C. M. Sperberg-McQueen. Extensible markup language (XML) 1.0 (second edition). `http://www.w3.org/TR/REC-xml`, October 2000.
6. G. D. Brown. IHTML 2: Design and Implementation. In W. W. Wadge, editor, *Proceedings of the 11th International Symposium on Languages for Intensional Programming*, pages 1–13, 1998.
7. Sudarshan S. Chawathe. Describing and manipulating XML data. *Bulletin of the IEEE Computer Society Technical Committee on Data Engineering*, 22(3):3–9, September 1999.
8. James Clark. XSL Transformations (XSLT). `http://www.w3.org/TR/xslt`, 1999.
9. M. Fernández, W.-C. Tan, and D. Suciu. SilkRoute: Trading between Relations and XML. *Computer Networks*, 33(1–6):723–745, 2000.
10. M. Gergatsoulis, Y. Stavrakas, and D. Karteris. Incorporating dimensions to XML and DTD. *It will be presented at International Conference on Database and Expert Systems Applications (DEXA' 01), Munich, Germany September, 2001.*
11. T. Mitakos, M. Gergatsoulis, Y. Stavrakas, and E. V. Ioannidis. Representing time-dependent information in multidimensional XML. *Proc. of the 23rd Int. Conf. "Information Technology Interfaces" (ITI'01), Pula, Croatia, June 2001.*
12. J. Plaice and W. W. Wadge. A New Approach to Version Control. *IEEE Transactions on Software Engineering*, 19(3):268–276, 1993.
13. Y. Stavrakas, M. Gergatsoulis, and T. Mitakos. Representing context-dependent information using Multidimensional XML. In J. Borbinha and T. Baker, editors, *Research and Advanced Technology for Digital Libraries, 4th European Conference ECDL'2000*, Lecture Notes in Computer Science (LNCS) 1923, pages 368–371. Springer-Verlag, 2000.
14. Y. Stavrakas, M. Gergatsoulis, and P. Rondogiannis. Multidimensional XML. In P. Kropf, G. Babin, J. PLaice, and H. Unger, editors, *Distributed Communities on the Web, Third International Workshop (DCW'2000)*, Lecture Notes in Computer Science (LNCS) 1830, pages 100–109. Springer-Verlag, 2000.
15. D. Suciu. An overview of semistructured data. *SIGACT News*, 29(4):28–38, December 1998.
16. W. W. Wadge, G. D. Brown, M. C. Schraefel, and T. Yildirim. Intensional HTML. In *Proceedings of the Fourth International Workshop on Principles of Digital Document Processing (PODDP '98)*, Lecture Notes in Computer Science (LNCS) 1481, pages 128–139. Springer-Verlag, March 1998.
17. T. Yildirim. Intensional HTML. Master's thesis, Department of Computer Science, University of Victoria, 1997.

An Abstract Database Machine for Cost Driven Design of Object-Oriented Database Schemas

Joachim Biskup and Ralf Menzel

Universität Dortmund, 44221 Dortmund, Germany
{biskup, menzel}@ls6.cs.uni-dortmund.de

Abstract The process of designing an object-oriented database schema consists of several phases. During the phase of abstract logical formalisation one of many possible abstract object-oriented database schemas must be chosen. This choice can be driven by the costs of the ultimately implemented schema: How much space is needed? How long does it take to compute queries and updates including enforcement of semantic constraints? Because abstract logical formalisation is done independently of an actual database management system, we need an abstract database machine. Queries and updates are formulated as programs for this database machine. Such programs are composed of steps which are connected by channels for typed streams of value lists. In each step, a basic or compound operation is executed, accepting input streams and further parameters, delivering output streams for subsequent steps, and accessing the persistent database state. The abstract database machine is designed to meet two goals: to be expressive enough to implement queries and updates, as considered for schema design, and to be simple enough to allow cost estimations.

1 Introduction

A database schema should have at least the following two properties. It should enable the user to represent all aspects (of the world) that are semantically relevant for the application, and it should be a syntactically valid schema for the given database system. A *good* database schema has additional properties. Many of these are given as heuristics, which are indications for a conceptually good schema [3]. Our main concern is the investigation of costs that a schema entails.

What do we mean when we say costs? First there is the *space* needed for storing the application data. Additionally there is space overhead for access structures added to the conceptual schema. These are the space costs that we want to directly exploit for schema design. Second there are *time* costs. The choice of the schema influences how long it takes the database system to answer *queries* or execute *updates*. The cost analysis should also include the costs for *maintenance* of the aforementioned access structures. There are two aspects to *semantic constraints*. On the one hand they can cause additional costs for checking them. On the other hand they can be employed to optimise queries or updates, and algorithms for database operations can utilise them to gain efficiency.

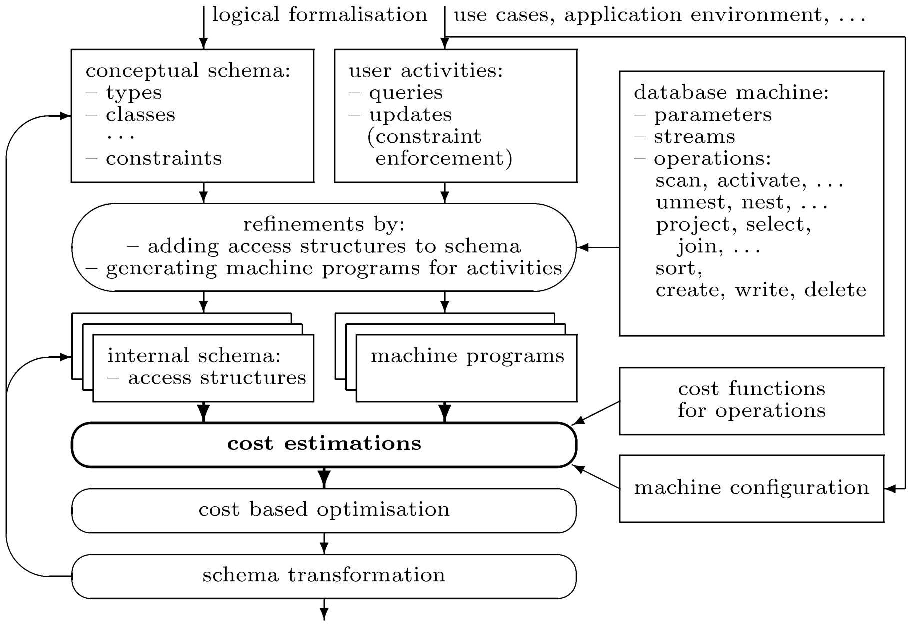


Fig. 1. Outline of cost driven database schema design

For the design of an object-oriented database schema we advocate a three-step methodology [5]. First, 'semantic modelling' analyses the requirements of the application. Its result can be represented by an entity-relationship schema or a UML class diagram. Semantic modelling is done to ensure that all semantically relevant aspects are captured. Second, 'abstract logical formalisation' transforms the initial schema into an abstract object-oriented schema. Then this abstract schema can be transformed in order to improve it. Third, 'concrete class declarations' gives the schema for the underlying object-oriented database system.

One way to measure costs would be by running some tests after we have arrived at a concrete object-oriented database system. If we deemed the measured costs too high, we probably would have to go back to our abstract object-oriented database schema and try to improve it. This could be a tedious process. In this paper we lay the foundations for a cost driven design which refines the second step of the full design methodology and includes a general cost model. The overall structure of this refinement is visualised in Fig. 1 and can be roughly outlined as follows: The conceptual part of an abstract object-oriented schema is given by describing types, classes and their hierarchy and semantic constraints. For our cost model we complement this *conceptual* schema with an *internal* schema, that lists the utilised access structures. There is a choice of different internal schemas for every conceptual schema. In part this reflects the different underlying database systems and the choices they give for implementing the conceptual schema.

The different *space* costs of an abstract schema can be estimated given the conceptual schema and a corresponding internal schema, and additionally, a collection of queries and updates which represent the anticipated user activities. For the analysis of *time* costs we describe an abstract object-oriented database machine and cost functions for its operations. Queries and updates are expressed as programs for the database machine. And the cost functions are steered by cost parameters. Most of the cost parameters are estimates for a probable database instance. Some of the parameters describe the underlying database system. Then, by a step by step analysis of the machine programs, we can estimate the time costs of the respective queries and updates.

There are several investigations considering costs while designing databases [13,16,17,19,20]. Of particular interest are the following. Bertino and Foscoli [2] present a model to characterise topologies of object references in object-oriented databases. They present a set of parameters that are similar to the cost parameters that we use to steer our cost functions. Vance [21] presents a preliminary design for an abstract query execution language. This is similar to our abstract database machine. Both approaches focus on queries and don't pay much attention to updates, constraint maintenance and access structures. Yao [23] compares different query evaluation algorithms for relational databases by breaking them into simple access operations. We seize this approach for our abstract object-oriented database machine. In a sense our abstract database machine describes an object algebra, where the operations are close to typical operations of a real database system. An object algebra is given by Osborn [14]. Velez, Bernard and Darnis [22] and Shekita [15] discuss implementation details of object-oriented database systems. Updates change the state of the persistent database. We use programs for an abstract database machine to represent updates, that must respect the semantic constraints. We outline a language for such programs, but we do not discuss how to actually write such programs. The latter task has been investigated by others [10,11,12]. Jagadish and Qian [8] present a constraint compilation scheme, that can be used to transform declarative semantic constraints into an efficient representation that permits localised processing. Gupta and Widom [7] show how to locally verify global integrity constraints.

This introduction is completed by introducing our running example and some formal notation. In Sect. 2 we sketch our object model, which is the basis for conceptual and internal schemas. Our main contribution, an abstract database machine, is presented in Sect. 3, where we select a representative collection of database operations that basically work on streams of value lists. Section 4 outlines the cost evaluation, which is based on the abstract database machine, and which can be exploited for the cost driven schema design.

We use a simple running example for illustration. It pictures a company with several departments. Every department has a number, a name and a location. In the departments work employees. Each employee has a name and a salary. Among the employees are some that manage others. These managers are given a budget to administer.

N is the set of natural numbers (without 0). We write $f : X \overset{\text{p}}{\rightarrow} Y$ to express that f is a partial function. A list l can be given by enumerating its elements, $\langle x_1, \ldots, x_n \rangle$, and its length, n, is denoted by $\|l\|$. Sometimes it is useful to be able to apply an operation f to every element of a list l. This is denoted by forall(l, f).

2 The Object Model

There is no such thing as *the* object-oriented model. One popular approach is 'The Object Data Standard: ODMG' [6]. Another related development is the emergence of object-relational databases as advocated by Stonebraker [18] and adopted in part by the latest SQL standard [1]. We will now describe the object-oriented model that we are going to use. It contains mainly the essential features commonly found in object-oriented data models like ODMG or F-Logic [9]. If you know F-Logic you will see that our model has explicit types and specifically special literal types. If you know ODMG, you can see that we have left out many of its possible data structures and just use sets and tuples. Figure 2 shows one possible abstract formalisation of our example database.

First, there is a *literal domain declaration.* It contains the *literal types*, $\mathcal{T}_{\text{Lit}}$, the *literals*, L, and a *literal domain assignment*, $\iota_{\text{Lit}} : \mathcal{T}_{\text{Lit}} \rightarrow \mathcal{P}L$, that yields an extension for every literal type. Over a literal domain declaration a *database schema* can be given. A schema declares a set of *object types*, $\mathcal{T}_{\text{Obj}}$, *classes*, $\mathcal{C}$, *attributes*, $\mathcal{A}$, a *class hierarchy*, $\trianglelefteq \subseteq \mathcal{C} \times \mathcal{C}$, that is a partial order of the classes, a *class type assignment*, $\gamma : \mathcal{C} \rightarrow \mathcal{T}_{\text{Obj}}$, that gives an object type for every class, an *attribute type assignment*, $\sigma : \mathcal{T}_{\text{Obj}} \times \mathcal{A} \rightarrow \mathcal{T}_{\text{Val}}$, that gives a value type for every pair of object type and attribute, and *semantic constraints*, SC.

Further components of a schema can be derived from the given ones. From every object type, t, an *object identifier type*, t_`OID`, is constructed, giving the set of object identifier types, $\mathcal{T}_{\text{OID}}$. The union of literal types and object identifier types is called *scalar types*, $\mathcal{T}_{\text{Scalar}}$. For every scalar type, t, a *set type*, t_`Set`, is constructed, giving the set of set types, $\mathcal{T}_{\text{Set}}$, that inherently consists of the literal set types, $\mathcal{T}_{\text{LitSet}}$, and the object identifier set types, $\mathcal{T}_{\text{OIDSet}}$. The special type $\top$ can be used, when no specific type is required. The union of all scalar types, set types plus the special type $\top$ is called *value types*, $\mathcal{T}_{\text{Val}}$. Then, the set of all *types*, $\mathcal{T}$, is the union of the set of value types and the set of object types.

Finally, a *specialisation hierarchy*, $\leq \subseteq \mathcal{T} \times \mathcal{T}$, is defined as a partial order of the types: Any type is more special then the type $\top$, an object type is more special than another object type, if all attribute types of the first are more special than the attribute types of the second. This is canonically extended from object types to object identifier types, and then from scalar types to set types. An object identifier type is more special than another object identifier type, if the first corresponds with an object type that is more special than the object type that the second object identifier type corresponds with. Similarly, a set type is more special than another set type, if the first corresponds with a scalar type that is more special than the scalar type that the second set type corresponds with.

Based on a database schema, a *database instance* can be given. A database instance contains *object identifiers*, I, an *attribute value assignment*, $\alpha : I \times \mathcal{A} \overset{\mathrm{p}}{\to} V$, that gives a value for *some* pairs consisting of an object identifier and an attribute, a *direct class extension assignment*, $\chi_0 : \mathcal{C} \to \mathcal{P}I$, that yields a set of object identifiers for every class, and an *object type extension assignment*, $\omega : \mathcal{T}_{\mathrm{Obj}} \to \mathcal{P}I$, that yields a set of object identifiers for every object type.

Again, we can derive further components of an instance from the given ones. The set of all *values*, V, contains all object identifiers, all literals, all sets of object identifiers and all sets of literals. The *class extension assignment*, $\chi : \mathcal{C} \to \mathcal{P}I$, yields a set of object identifiers that is the union of all direct extensions of the respective class and its subclasses. Finally, there is the *type extension assignment*, $\iota : \mathcal{T}_{\mathrm{Val}} \to \mathcal{P}V$, that yields a set of values for every value type. It is defined as a canonical extension of the literal domain assignment and the object type extension assignment.

An *internal schema* consists of a set of *access structure declarations*, $X \subseteq \mathcal{C} \times \mathcal{A}$, that lists all access structures by giving a class and an attribute of the class, (c, A), for each access structure. An access structure characterised by (c, A) can be used to retrieve objects of class c that have some given index value for attribute A. If the type for A is a scalar type the access structure returns all objects that have the index value as attribute value for the attribute A. If the type for A is a set type the access structure returns all objects that contain the index value in their attribute value for attribute A.

3 Abstract Database Machine

The rough structure of the abstract database machine is shown in Fig. 3. The machine works by evaluating programs. A machine program is composed of steps which are connected by channels. Each of these steps contains an operator with the specification of its input and output. Data transfer from and to secondary memory is encapsulated by the operators 'scan', 'activate', 'write', 'delete' and 'access'. The data flow within the main memory is represented as input and output arguments. Among the different types for arguments there is one pivotal type, that is used for data passed from one operation to another using a channel. This data transfer within the database machine is represented by streams.

3.1 Streams

A *stream* is a sequence of *value lists*. The values in the value lists are numbered starting with 1. The number for a value is called its *position*.

All value lists that are part of the same stream must have the same type. This *stream type*, $(t_1, \ldots, t_n)$, gives the number of values in the value lists, n, and the types for every position, $t_k \in \mathcal{T}_{\mathrm{Val}} \setminus \{\top\}$.

The attribute value assignment of our object model is a partial function. It can be use to assign null values to attributes. We must therefore provide means

We use the following literal domain declaration:
literal types: $\mathcal{T}_{\text{Lit}} = \{\texttt{integer}, \texttt{string}\}$
literal domain assignment: $\iota_{\text{Lit}}(\texttt{integer}) = \{\ldots, -1, 0, 1, 2, 3, \ldots\}$
$\iota_{\text{Lit}}(\texttt{string}) = \{\text{' '}, \text{'a'}, \text{'b'}, \ldots, \text{'aa'}, \ldots\}$

The abstract logical formalisation is done as the following database schema:
object types: $\mathcal{T}_{\text{Obj}} = \{\text{department}, \text{employee}, \text{manager}\}$
classes: $\mathcal{C} = \{\text{Department}, \text{Employee}, \text{Manager}\}$
attributes: $\mathcal{A} = \{\text{number}, \text{name}, \text{location}, \text{workers}, \text{salary}, \text{boss}, \text{budget}\}$
class hierarchy: $\trianglelefteq = \{(\text{Manager}, \text{Employee})\}$
class type assignment: $\gamma(\text{Department}) = \text{department}$, $\gamma(\text{Employee}) = \text{employee}$,
$\gamma(\text{Manager}) = \text{manager}$
attribute type assignment: $\sigma(\text{department}, \text{number}) = \texttt{integer}$, $\sigma(\text{department}, \text{name}) = \texttt{string}$,
$\sigma(\text{department}, \text{location}) = \texttt{string}$, $\sigma(\text{department}, \text{workers}) = \text{employee\_}\texttt{OID\_Set}$,
$\sigma(\text{employee}, \text{name}) = \texttt{string}$, $\sigma(\text{employee}, \text{salary}) = \texttt{integer}$, $\sigma(\text{employee}, \text{boss}) = \text{manager\_}\texttt{OID}$,
$\sigma(\text{manager}, \text{name}) = \texttt{string}$, $\sigma(\text{manager}, \text{salary}) = \texttt{integer}$, $\sigma(\text{manager}, \text{boss}) = \text{manager\_}\texttt{OID}$,
$\sigma(\text{manager}, \text{budget}) = \texttt{integer}$

A database instance for the given database schema might look as follows:
object identifiers: $I = \{i_1, i_2, \ldots, i_{15}\}$
attribute value assignment: $\alpha(i_1, \text{number}) = 10$, $\alpha(i_1, \text{name}) = \text{'Research'}$,
$\alpha(i_1, \text{location}) = \text{'London'}$, $\alpha(i_1, \text{workers}) = \{i_4, i_5, i_6, i_7\}$
$\alpha(i_2, \text{number}) = 20$, $\alpha(i_2, \text{name}) = \text{'Marketing'}$, $\alpha(i_2, \text{location}) = \text{'Paris'}$,
$\alpha(i_2, \text{workers}) = \{i_8, i_9, i_{10}\}$
...
$\alpha(i_{14}, \text{name}) = \text{'Muller'}$, $\alpha(i_{14}, \text{salary}) = 17\,000$, $\alpha(i_{14}, \text{budget}) = 200\,000$, $\alpha(i_{14}, \text{boss}) = i_{15}$
$\alpha(i_{15}, \text{name}) = \text{'Goodwin'}$, $\alpha(i_{15}, \text{salary}) = 32\,000$, $\alpha(i_{15}, \text{budget}) = 700\,000$
direct class extension assignment: $\chi_0(\text{Department}) = \{i_1, i_2, i_3\}$
$\chi_0(\text{Employee}) = \{i_4, i_5, i_6, i_8, i_9, i_{11}, i_{12}, i_{13}\}$
$\chi_0(\text{Manager}) = \{i_7, i_{10}, i_{14}, i_{15}\}$
object type extension assignment: $\omega(\text{department}) = \{i_1, i_2, i_3\}$
$\omega(\text{employee}) = \{i_4, i_5, i_6, i_7, i_8, i_9, i_{10}, i_{11}, i_{12}, i_{13}, i_{14}, i_{15}\}$
$\omega(\text{manager}) = \{i_7, i_{10}, i_{14}, i_{15}\}$

Finally, we choose to have access structures for the class 'Department' on the attributes 'number' and 'workers', for the class 'Employee' on the attributes 'name' and 'boss', and for the class Manager on the attribute 'name':
$X = \{(\text{Department}, \text{number}), (\text{Department}, \text{workers}), (\text{Employee}, \text{name}), (\text{Employee}, \text{boss}), (\text{Manager}, \text{name})\}$

Fig. 2. A formalisation of the example database

to represent null values in our value lists. For that purpose we employ the special value $\perp$ that is different from any value already in V. This value is added to the set of all values, V, and to all type extensions $\iota(t)$. The attribute value assignment returns $\perp$ for any attribute of $\perp$, that is, $\alpha(\perp, A) = \perp$ for all $A \in \mathcal{A}$.

A *value list* is a tuple of values, $(v_1, \ldots, v_n)$, $v_k \in V$. A value list $(v_1, \ldots, v_n)$ *is of stream type* $(t_1, \ldots, t_{n'})$, if $n = n'$ and $\forall k : v_k \in \iota(t_k)$.

A *stream* is a sequence of value lists, $\langle(v_{1,1}, \ldots, v_{1,n}), \ldots, (v_{m,1}, \ldots, v_{m,n})\rangle$. A stream *is of a stream type*, if all its value lists are of the stream type.

Some operations require that their input streams are sorted. For that we use *sortings* of streams. A sorting of a stream is given as a list of positions, $(k_1, \ldots, k_l)$. This list describes a lexicographic order: For every pair of value lists $(v_1, \ldots, v_n)$ and $(v'_1, \ldots, v'_n)$, where $(v_1, \ldots, v_n)$ comes before $(v'_1, \ldots, v'_n)$ in the stream s and there is a position given by the sorting where they differ, $\{x \mid v_{k_x} \neq v'_{k_x}\} \neq \emptyset$, for the most significant differing position, $a = \min\{x \mid v_{k_x} \neq v'_{k_x}\}$, the value of the first value list must be smaller than the value of the second value list,

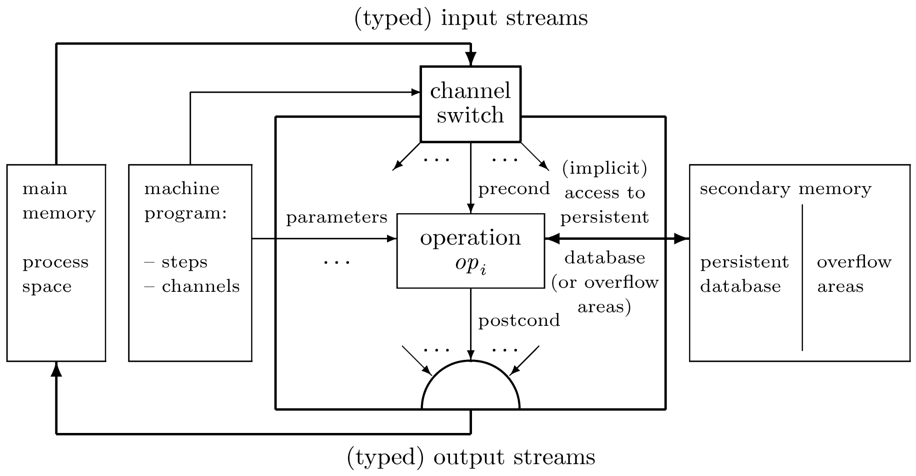


Fig. 3. The abstract database machine

$v_{k_a} \leq v'_{k_a}$. We often write 'The stream s is sorted on $(k_1, \ldots, k_l)$' instead of 'The sorting of s is $(k_1, \ldots, k_l)$'. Note, that the sorting can be done for all value types.

A stream s of type $(t_1, \ldots, t_n)$ is *completely sorted*, iff it is sorted on some $(k_1, \ldots, k_n)$. That is $\{k_1, \ldots, k_n\} = \{1, \ldots, n\}$.

3.2 Operations

Now, that we have defined streams, we can go on to describe the individual operations of the abstract database machine. First we define a set of basic operations. After that we define some compound operations that use the basic operations as building blocks. The *unabridged* description of each operation begins with the name and a synopsis of the operation. Here we mention important aspects of an operation: if it accesses the persistent database state, if it might generate duplicates in its output, or if it needs its entire input stream to properly operate. After that we first list the *parameters*, all the arguments that aren't input streams coming from other operations. Second, come *input streams* together with any precondition that must be fulfilled. Note, that all the operations that require a sorted input, have a name with a zero as index. After the description of the input arguments is the description of possible *output streams*. If there are output streams, we give their types, any postconditions they fulfil and define how they can be calculated. Finally, for operations that modify the persistent database state, we describe this *database instance update*.

You can find the unabridged descriptions of all operations in a technical report [4], that we've written. In this paper we only give a complete description for some of the operations. Often we restrict the description to the synopsis of the operation. Table 1 summarises some interesting properties of the operations.

Table 1. Operations of the database machine

name	number of input streams	number of output streams	access to persistent db state	may generate duplicates	sorted input required	needs entire input stream for operation
const	—	1	—	yes	n/a	n/a
duplicate	1	2	—	—	—	—
scan	—	1	read	—	n/a	n/a
activate	1	1	read	—	—	—
unnest	1	1	—	yes	—	—
nest_0	1	1	—	—	yes	—
project	1	1	—	yes	—	—
unique_0	1	1	—	—	yes	—
select	1	1	—	—	—	—
sort	1	1	—	yes	—	yes
product	2	1	—	—	—	yes
unionall	2	1	—	yes	—	—
create	1	1	—	—	—	—
write	1	—	write	n/a	—	—
delete	1	—	write	n/a	—	—
union_0	2	1	—	—	yes	—
join_0	2	1	—	—	yes	—
access	1	1	read	—	—	—

Basic Operations. The first two basic operations, 'const' and 'duplicate', are there for technical reasons. With them it is possible to write machine programs where every input stream is the output of another operation and where every output stream is only used once.

The 'const' operation gives a constant stream. It can be used for queries and updates that use constant values.

'duplicate' takes it input stream and returns it twice. It is used whenever the output of an operation must be processed in two different ways.

'scan' takes a class and generates the stream of object identifiers of the class. It is typically one of the first operations in a program, because it is the only operation that reads data from the persistent database state and requires no input stream to do this.

'activate' loads the objects for all object identifiers at a given position. This operation is most often employed to navigate references from one class to another. Besides 'scan' it is the only other basic operation that reads data from the persistent database state.

- parameter: a position k.
- input stream: a stream s of type $(\ldots, t_k, \ldots)$, where t_k is an object identifier type, i.e., $t_k \in \mathcal{T}_{\text{OID}}$.
- output stream: a stream of type $(\ldots, t_k, \ldots, t'_1, \ldots, t'_n)$, where $t'_1, \ldots, t'_n$ are the types of the defined attributes of t_k, i.e., $t'_l = \sigma(t_k, A_l)$ with $\{A_1, \ldots, A_n\} = \text{attr}(t_k)$ and $l < l' \Rightarrow A_l <_{\mathcal{A}} A_{l'}$ for a globally given total order $<_{\mathcal{A}}$.

The output stream is sorted like the input stream.
activate(s,k) = forall(s,f_k), where $f_k((\ldots,v_k,\ldots)) = \langle(\ldots,v_k,\ldots,v'_1,\ldots,v'_n)\rangle$ with $v'_l = \alpha(v_k, A_l)$.

'unnest' is used to inspect the elements of a set. Every value list of the input stream generates a sequence of value lists in the output stream by replacing the set at the specified position by the values in the set, one after another. 'unnest' is one of the operations the might generate duplicate value lists in its output.

- parameters: a position k.
- input stream: a stream s of type $(\ldots,t_k,\ldots)$, where t_k is a set type, i.e., $t_k \in \mathcal{T}_{\text{Set}}$.
- output stream: a stream of type $(\ldots,t'_k,\ldots)$, where t_k is the set type of t'_k, i.e., $t_k = t'_k$_`Set`.
  The output stream is sorted like the input stream.
  unnest(s,k) = forall(s,f_k), where $f_k((\ldots,v_k,\ldots)) = \langle(\ldots,v'_{k,1},\ldots),\ldots,(\ldots,v'_{k,l},\ldots)\rangle$ with $\{v'_{k,1},\ldots,v'_{k,l}\} = v_k$ and $l = \|v_k\|$.

'nest$_0$' replaces each sub-sequence of elements of a sorted input stream that are identical except for one given position with one value list in the output stream. For the specified position the resulting value list has a set of all values that the value lists of the input stream have at that position.

- parameter: a position k.
- input stream: a stream s of type $(t_1,\ldots,t_k,\ldots,t_n)$, where t_k is a scalar type, i.e., $t_k \in \mathcal{T}_{\text{Scalar}}$.
  The stream s must sorted on some $(k_1,\ldots,k_m)$, with $\{k_1,\ldots,k_{n-1}\} = \{1,\ldots,k-1,k+1,\ldots,n\}$.
- output stream: a stream of type $(t_1,\ldots,t'_k,\ldots,t_n)$, where t'_k is the set type of t_k, i.e., $t'_k = t_k$_`Set`.
  The output stream is sorted like the input stream.

$$\text{nest}_0(s,k) = \begin{cases} \langle\rangle & \text{if } s = \langle\rangle \\ \langle(v_1,\ldots,\{v_k\},\ldots,v_n)\rangle & \text{if } s = \langle(v_1,\ldots,v_k,\ldots,v_n)\rangle \\ \langle x_1\rangle + \text{nest}_0(\langle x_2,\ldots\rangle) & \text{if } s = \langle x_1,x_2,\ldots\rangle \wedge \neg(x_1 =_{\bar{k}} x_2) \\ \text{add}_k(x_1, \text{nest}_0(\langle x_2,\ldots\rangle)) & \text{if } s = \langle x_1,x_2,\ldots\rangle \wedge x_1 =_{\bar{k}} x_2 \end{cases}$$

with $(v_1,\ldots,v_n) =_{\bar{k}} (v'_1,\ldots,v'_n) \iff \forall l \neq k : v_l = v'_l$ and where $\text{add}_k((v_1,\ldots,v_k,\ldots,v_n),\langle(v'_1,\ldots,v'_k,\ldots,v'_n),x'_3,\ldots\rangle) = \langle(v'_1,\ldots,v'_{k-1},v'_k \cup \{v_k\},v'_{k+1},\ldots,v'_n),x'_3,\ldots\rangle$.

'project' disposes of parts of the value lists that are no longer needed. There is no removal of duplicates. It is typically used at the end of a program to select only the required values, or before one of the operations that operate on the entire stream, like 'sort' or 'product', to reduce the amount of main memory these operations might need.

'unique$_0$' removes duplicates in a sorted stream. There are several operation that might generate duplicates in their output streams even though their input streams didn't contain duplicates. Because only adjacent value lists in the input stream are inspected by 'unique$_0$' the input stream must be completely sorted.

'select' removes all value lists that do not satisfy a given predicate. It is typically used in the midst of a program to select data that satisfies some requirements. We refrain from further specifying different types of selection predicates. But comparing a value in a value list with a constant value or another value in the value list are probably the most common cases.

'sort' generates as output a sorted version of its input. The sorting of the output is the one specified as parameter of the operation. The sort operation is particularly important, because several other operations can only be used on sorted input. It should be evident that 'sort' needs the entire input stream for operation.

- parameters: a non-empty list of distinct positions $(k_1, \ldots, k_l)$, with $k_m = k_{m'} \Rightarrow m = m'$.
- input stream: a stream s of type $(t_1, \ldots, t_n)$, where $\{k_1, \ldots, k_l\} \subseteq \{1, \ldots, n\}$.
- output stream: a stream of type $(t_1, \ldots, t_n)$.
  The output stream is sorted on $(k_1, \ldots, k_l)$.

'product' takes two input streams and yields their Cartesian product. Most programs probably will not contain a product operation. We provide it for completeness and as a means to define the rather important join operation. 'product' needs the entire input stream for operation.

'unionall' takes two input streams and appends the second to the first. It might generate duplicates in its output. When we assume that we normally are not interested in duplicates, we will probably try to use 'union$_0$' instead of 'unionall'.

'create' generates new object identifiers. When we want to create objects we need the values for their attributes and new object identifiers. The attribute values can be created by 'const' or read from the persistent database state. But for the new object identifier we need 'create'. To finally write the objects to the persistent database state the write operation must be used.

- parameter: an object identifier type t, i.e. $t \in \mathcal{T}_{\text{OID}}$.
- input stream: a stream s of type $(t_1, \ldots, t_n)$.
- output stream: a stream of type $(t_1, \ldots, t_n, t)$.
  The output stream is sorted like the input stream.

'write' can be used to insert and update objects. The objects that are written are extracted from the input stream, that contains in each value list the attributes and object identifier for one object. When inserting new objects the object identifiers must have been created by 'create'. When updating objects the object identifiers have been read from the persistent database state. The written objects are placed in the direct extension of the specified class. Next to 'delete' this is the only operation that changes the persistent database state.

- parameters: a class name c, a position l and a function $f : \text{attr}(\gamma(c)) \to \mathbf{N}$,
- input stream: a stream s of type $(t_1, \ldots, t_n)$, where $\bigcup_{A \in \text{attr}(\gamma(c))} f(A) \subseteq \{1, \ldots, n\}$ and $t_l \leq \gamma(c)$ and $t_{f(A)} \leq \sigma(c, A)$.
- output stream: none.

- database instance update:
  - $I := I \cup J \cup \bar{I}$
  - $\alpha(v_l, A) := v_{f(A)}$, for $(v_1, \ldots, v_n) \in s$ and $A \in \mathrm{attr}(\gamma(c))$
  - $\chi_0(c) := \chi_0(c) \cup J$
  - $\omega(t) := \omega(t) \cup J$ for all $t \geq t_l$

  where $J = \{v_l \mid (v_1, \ldots, v_n) \in s\}$ and $\bar{I} = \{v_{f(A)} \mid (v_1, \ldots, v_n) \in s \wedge t_{f(A)} \in \mathcal{T}_{\mathrm{OID}} \wedge A \in \mathrm{attr}(\gamma(c))\}$.

'delete' removes objects from class extensions. All the objects that are given by the object identifiers in the input stream at the specified position are removed from the direct extension of the specified class and its subclasses. Besides 'write' this is the only other operation that updates the persistent database state.

Compound Operations. The following operations can be expressed by means of the basic operations. That means that they do not extend the expressiveness. They are provided, because they typically entail smaller costs than the costs that are caused by the equivalent composition of basic operations.

'union$_0$' takes two equally sorted input streams and unites them. Duplicates are removed. In contrast to its counterpart, the basic operation 'unionall', the result of 'union$_0$' is sorted.

'join$_0$' takes two sorted input streams and joins them. This is a standard equi-join for one join value. It is not only useful to compute a join for literal values but also for object identifiers.

- parameters: a position k and a position l.
- input streams: a stream s_1 of type $(t_1, \ldots, t_k, \ldots, t_n)$ sorted on $(k, k_1, \ldots, k_p)$ and a stream s_2 of type $(t'_1, \ldots, t'_l, \ldots, t'_m)$ sorted on $(l, \ldots)$.
- output stream: a stream of type $(t_1, \ldots, t_k, \ldots, t_n, t'_1, \ldots, t'_l, \ldots, t'_m)$ sorted on $(k, k_1, \ldots, k_p)$.
  $\mathrm{join}_0(s_1, s_2, k, l) = \mathrm{select}(\mathrm{product}(s_1, s_2), \mathrm{p}_{k,n+l})$ where $\mathrm{p}_{k,n+l}(v_1, \ldots, v_{n+m}) \iff v_k = v_{n+l}$.

'access' uses an access structure to retrieve the objects for the given key values. For the values at the specified position in the input stream all objects of the specified class that have these values for the specified attribute are loaded and combined with the respective value lists. Only value lists of the input stream where matching objects can be found reappear in the output stream. Next to the basic operations 'scan' and 'activate', it is the third operation the reads data from the persistent database state.

- parameters: an access structure declaration $(c, A) \in X$ and a position k.
- input stream: a stream s of type $(t_1, \ldots, t_k, \ldots, t_n)$, where t_k is the type of the keys of the access structure, i.e. $t_k = \mathrm{keytype}((c, A))$.
- output stream: a stream of type $(t_1, \ldots, t_n, t\_\mathtt{OID}, t'_1, \ldots, t'_m)$, where $t = \gamma(c)$ and the t'_l are the types of the defined attributes of t as describe for 'activate'. The output stream is sorted like the input stream.

$x_1 := \mathit{const}(\langle(\text{'Jones'})\rangle, (\texttt{string}))$
$x_2 := \mathit{access}(x_1, (\text{Manager}, \text{name}), 1)$
$x_3 := \mathit{access}(x_2, (\text{Employee}, \text{boss}), 2)$
$x_4 := \mathit{project}(x_3, f)$ with $f(1) = 8$

x_1: $(\texttt{string})$
x_2: $(\texttt{string}, \text{manager\_}\texttt{OID}, \texttt{string}, \texttt{integer}, \text{manager\_}\texttt{OID}, \texttt{integer})$
x_3: $(\texttt{string}, \text{manager\_}\texttt{OID}, \texttt{string}, \texttt{integer}, \text{manager\_}\texttt{OID}, \texttt{integer},$
$\text{employee\_}\texttt{OID}, \texttt{string}, \texttt{integer}, \text{manager\_}\texttt{OID})$
x_4; (string) x_1: $\langle(\text{'Jones'})\rangle$

x_2: $\langle(\text{'Jones'}, i_7, \text{'Jones'}, 18\,000, i_{15}, 180\,000)\rangle$
x_3: $\langle(\text{'Jones'}, i_7, \text{'Jones'}, 18\,000, i_{15}, 180\,000, i_4, \text{'Smith'}, 12\,000, i_7),$
$(\text{'Jones'}, i_7, \text{'Jones'}, 18\,000, i_{15}, 180\,000, i_5, \text{'Kline'}, 14\,500, i_7),$
$(\text{'Jones'}, i_7, \text{'Jones'}, 18\,000, i_{15}, 180\,000, i_6, \text{'Adams'}, 16\,000, i_7)\rangle$
x_4: $\langle(\text{'Smith'}), (\text{'Kline'}), (\text{'Adams'})\rangle$

Fig. 4. A machine program for computing the names of all employees managed by Jones, its respective stream types, and the actual streams for the example database.

3.3 Machine Program

A program for the database machine consists of one or more steps. Each step designates one operation and describes the input and output of the operation. For this purpose, streams are represented by program variables of the form x_k, called *channels*. Every input stream for an operation is given as a channel that stands for the output stream of an earlier step.

A *machine program step* is denoted as $op(args)$, for operations without an output stream, as $x_k := op(args)$, for operations with one output stream, or as $(x_k, x_l) := op(args)$, for operations with two output streams. Here x_k denotes a channel, *op* is an operator and *args* is a comma separated list of suitable arguments. All the parameters of an operation must be provided as concrete values. If the operation requires an input stream, the stream is given as a channel x_l.

A *machine program* is a sequence of machine program steps that satisfies the following requirements: The preconditions of the operators in all steps are satisfied. All arguments are of valid types in particular. Each channel is used as output by exactly one step. (The two outputs of 'duplicate' must be two different channels.) Each channel is used as input by at most one step. Each channel that is used as input by a machine program step is used as output by an earlier step.

Note, that every program can be represented by an ordered directed acyclic graph, where the nodes stand for program steps and the vertices for channels.

4 Cost Evaluation

The space costs can be determined without using the abstract database machine. So, for the sake of brevity, we only consider time costs. The time costs are evaluated by looking at the steps of the machine program one after the other. In this process all program steps and all channels in the program are adorned with *cost statements*. When we have a program step it contains an operation. For the purpose of cost evaluation we must provide *cost functions* for every operation. These cost functions are applied to the parameters of the current step,

Application dependent parameters:

- N_c: number of objects in the extension of class c.
- $N_{0,c}$: number of objects in the direct extension of class c.
- n_t: average number of blocks for an object of type t.
- $n_{(t_1,\ldots,t_n)}$: average number of blocks for a value list of type $(t_1, \ldots, t_n)$.
- $n_{\mathrm{u}:t}$: average number of elements in a set of type t.
- $n_{\mathrm{A}:c,A}$: average number of values for attribute A of class c.
- $\theta_{\mathrm{s}:c}$: average filling level of the object identifier access structure for class c.
- $\theta_{\mathrm{a}:c}$: portion of objects of class c that have be moved from their original second memory location.
- θ_p: selectivity of predicate p.
- $\theta_{\mathrm{d}:c}$: fraction of objects in class c that are only members of c or any subclasses of c.
- $\theta_{\mathrm{D}:c}$: correction factor for deletion of objects of class c.
- $f_{\mathrm{q}:(t_1,\ldots,t_n)}(l)$: average number of unique elements in a stream of type $(t_1, \ldots, t_n)$ of length l.

System dependent parameters:

- n_{a}: fraction of a block that is needed for the non-data part of an access structure per element of the stored set.
- n_{OID}: number of object identifiers that fit into one block.
- n_{mem}: size of memory that is reserved for sorting and similar operations.
- n_{files}: maximum number of open files for sorting.
- $\eta(n)$: number of block accesses required to locate an element using an access structure for a set with n elements.

Fig. 5. Some cost parameters used by the cost evaluation

to the types, sortings and the cost statements of the input channels. The results of this application are used as the cost statement of the step and as the cost statement for any output channels, respectively. After every step is adorned with a cost statement, the total cost of the machine program results by combining the cost statements of all program steps with an *aggregation function*. We call the result of the aggregation *total cost statement*. We use simple cost functions, that distinguish between access to blocks of secondary memory and accesses to values that reside in main memory. More elaborated cost functions are possible, but beyond the scope of this exposition.

Now we could look at the cost functions for each operation in detail. But for space limitation we refrain from doing so. Instead, in order to give an impression of the flavour of the cost functions, we only list the main cost parameters in Fig 5.

5 Conclusion

We presented an abstract database machine for object-oriented databases, and we outlined how to exploit this machine for refining a general database schema design methodology in order to take cost optimisation into account. The detailed cost model has been beyond the scope of this presentation. Current research is devoted to thoroughly study the impact of costs for the full process of schema design. We mainly have to investigate the impact of semantic constraints, how they suggest to declare access structure, how they influence the need for sorting, and how they can be enforced with low costs.

References

1. American National Standards Institute, 11 West 42nd Street, New York, NY, 10036. *ANSI/ISO/IEC 9075-2-1999, Information Technology – Database Languages – SQL – Part 2: Foundation (SQL/Foundation)*, Dec. 8, 1999.
2. E. Bertino and P. Foscoli. On modeling cost functions for object-oriented databases. *IEEE Transaction on Knowledge and Data Engineering*, 9(3):500–508, May/June 1997.
3. J. Biskup. Database schema design theory: Achievements and challenges. In S. Bhalla, editor, *Proc. of the 6th Intl. Conf. Information Systems and Management of Data*, number 1006 in LNCS, pages 14–44, Bombay, 1995. Springer.
4. J. Biskup and R. Menzel An abstract database machine for cost driven design of object-oriented database schema. Technical Report, Universität Dortmund, 2001. http://ls6-www.cs.uni-dortmund.de/issi/archive/literature/2001/Biskup_Menzel:2001.ps.gz
5. J. Biskup, R. Menzel, and T. Polle. Transforming an entity-relationship schema into object-oriented database schemas. In J. Eder and L. A. Kalinichenko, editors, *Advances in Databases and Information Systems, Moscow 95*, Workshops in Computing, pages 109–136. Springer-Verlag, 1996.
6. R. G. G. Cattell and D. K. Barry, editors. *The Object Data Standard: ODMG 3.0.* Morgan Kaufmann, 2000.
7. A. Gupta and J. Widom. Local verification of global integrity constraints in distributed databases. In P. Buneman and S. Jajodia, editors, *Proc. of the 1993 ACM SIGMOD Intl. Conf. on Management of Data*, pages 49–58, 1993.
8. H. V. Jagadish and Xiaolei Qian. Integrity maintenance in an object-oriented database. In Li-Yan Yuan, editor, *Proc. of the 18th Intl. Conf. on Very Large Data Bases*, pages 469–480, British Columbia, Canada, 1992.
9. M. Kifer, G. Lausen, and J. Wu. Logical foundations of object-oriented and frame-based languages. *Journal of the ACM*, 42(4):741–843, 1995.
10. U. Lipeck. Transformation of dynamic integrity constraints into transaction specifications. *Theoretical Computer Science*, 76:115–142, 1990.
11. W. W. McCune and L. J. Henschen. Maintaining state constraints in relational databases: A proof theoretic basis. *Journal of the ACM*, 36(1):46–68, 1989.
12. J.-M. Nicolas. Logic for improving integrity checking in relational databases. *Acta Informatica*, 18(3):227–253, 1982.
13. B. A. Nixon. Representing and using performance requirements during the development of information systems. In M. Jarke, J. Bubenko, and K. Jeffery, editors, *Advances in Database Technology—EDBT '94*, number 779 in LNCS, pages 187–200. Springer, 1994.
14. S. L. Osborn. Identity, equality and query optimization. In K. R. Dittrich, editor, *Advances in Object-Oriented Database Systems, 2nd Intl. Workshop on Object-Oriented Database Systems*, number 334 in LNCS, pages 346–351, Bad Münster am Stein-Ebernburg, Germany, Sept. 27–30, 1988. Springer-Verlag.
15. E. J. Shekita. *High-Performance Implementation Techniques For Next-Generation Database Systems.* PhD thesis, U. of Wisconsin, Madison, 1990.
16. M. Steeg. The conceptual database design optimizer CoDO – concepts, implementation, application. In B. Thalheim, editor, *Proc. of the 15th Intl. Conf. on Conceptual Modeling*, number 1157 in LNCS, pages 105–120, Cottbus, Germany, Oct. 7–10, 1996. Springer.

17. M. Steeg and B. Thalheim. A computational approach to conceptual database optimization. Technical report, BTU Cottbus, May 1995.
18. M. Stonebraker, P. Brown, and D. Moore. *Object-Relational DBMSs: Tracking the Next Great Wave*. Morgan Kaufmann, Second edition, 1999.
19. P. van Bommel. Experiences with EDO: An evolutionary database optimizer. *Data & Knowledge Engineering*, 13(1994):243–263, 1994.
20. P. van Bommel and T. P. van der Weide. Reducing the search space for conceptual schema transformation. *Data & Knowledge Engineering*, 8(1992):269–292, 1992.
21. B. Vance. An abstract object-oriented query execution language. In C. Beeri, A. Ohori, and D. E. Shasha, editors, *Proc. of the Fourth Intl. Workshop on Data Base Programming Languages – Object Models and Languages*, Workshops in Computing, pages 176–199, New York City, 30 Aug.–1 Sept. 1993. Springer.
22. F. Velez, G. Bernard, and V. Darnis. The O_2 object manager: An overview. In P. M. G. Apers and G. Wiederhold, editors, *Proc. of the 15th Intl. Conf. on Very Large Data Bases*, pages 357–366, Amsterdam, 1989.
23. S. Bing Yao. Optimization of query evaluation algorithms. *ACM Transactions on Database Systems*, 4(2):133–155, June 1979.

Author Index